UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

FINAL BACHELOR'S DEGREE PROJECT

Bachelor of Biomedical Engineering

SEGMENTATION AND TRACKING OF MITOCHONDRIAL MOTION IN NEURAL CULTURES

Report and Annexes

Author: Clàudia Serrano Amenós
Advisor: Raúl Benítez Iglesias

Call: June 2018

Resum

L'anàlisi manual del trànsit de mitocòndries en els axons neuronals és una tasca que requereix molt de temps i a la vegada és propensa a incloure errors degut al factor humà. Per aquest motiu, en els últims anys s'han realitzat diversos intents per crear sistemes que siguin capaços de fer l'anàlisi automàticament. Tot i així, cap dels programes resultants pot funcionar correctament per a qualsevol tipus d'imatge de trànsit mitocondrial.

L'objectiu d'aquest projecte és crear un programa que sigui capaç de fer un anàlisi automàtic del moviment mitocondrial en cultius neuronals, més concretament en imatges 2D amb una freqüència de mostreig baixa, que es van obtenir durant un estudi desenvolupat per la Fundació Sant Joan de Déu.

La finalitat principal del projecte és prendre un sistema automàtic que ja existia però no funcionava correctament, identificar els errors que conté i modificar-lo per tal d'obtenir un mètode robust que ofereixi resultats fiables. A més a més, s'ha inclòs una validació del programa final en referència als resultats obtinguts per un expert en el tema i un mètode ja publicat, per tal de garantir la qualitat del programa i els seus resultats.

Resumen

El análisis manual del tráfico de mitocondrias en los axones neuronales es una tarea que requiere mucho tiempo y a la vez es muy propensa a introducir errores humanos. Por esta razón en los últimos años se han realizado diversos intentos para crear sistemas que sean capaces de hacer este análisis de forma automática. Aun así, ninguno de los programas resultantes puede funcionar para cualquier tipo de imagen de tráfico mitocondrial

El objetivo de este proyecto es crear un programa que sea capaz de realizar un análisis automático del movimiento mitocondrial en cultivos neuronales, más concretamente en imágenes 2D con una frecuencia de muestreo muy baja que se sacaron durante un estudio desarrollado en la Fundació Sant Joan de Déu.

La finalidad principal del proyecto es considerar un sistema automático ya existente però que no funcionaba correctamente, identificar los errores que presenta y modificarlo para poder obtener un método robusto que ofrezca resultados fiables. Además, se ha incluido una validación con respecto a un experto y un método ya publicado para garantizar la calidad del programa y sus resultados.

Abstract

The manual analysis of mitochondria travelling along the neural is a very time consuming and prone to human error task. That's why in the recent years there have been many attempts at systems that can automatically do so. However, none of these attempts have offered programs that are valid for any type of mitochondrial trafficking images.

This project aims to create a program that is able to do automated analysis of mitochondrial movement on neural cultures, more precisely for 2D images with a low framerate that have been obtained during a research carried out in Fundació Sant Joan de Déu.

The project's main goal is to take an already existing automated system that wasn't working properly enough, identify its flaws and modify it in order to obtain a robust method that can offer reliable results. Moreover, the algorithm's validation against an expert knowledge and an already existing and published software have been included in order to guarantee the quality of the program and its results.

Acknowledgements

I would like to thank my advisor Raúl Benítez, for his help, motivation and guidance throughout the project. Also, I am very grateful for Azahara Civera in Sant Joan de Déu, for her endless patience during the image and biological interpretation phase. Also in Sant Joan de Déu, special thanks to Janet Hoenicka, for giving me the chance to take part in this collaboration. I would also like to show my appreciation to Àlex Vallmitjana, for his wise advice and support regardless of the large distance and time difference. Finally, I am particularly grateful to my lab colleagues, Carme, Vicent and Xavier, for their daily encouragement and assistance along the entire project.

Glossary

WT: Wild type specimen. Specimen that has not been genetically modified.

KO: Knockout specimen. Specimen that has been genetically modified.

CMT: Charcot-Marie-Tooth Syndrome

GLCM: Gray-Level Co-occurrence Matrix

GT: Ground Truth.

Index

Figure Index

Table Index

1. Introduction

We live in a world where technology is present in almost all disciplines and fields of work. For instance, the industry is becoming more and more automated every day and medicine is incorporating some groundbreaking machinery that is able to solve long-existing problems. The introduction of this technology into our daily life is not only helping our society avoid repetitive and time consuming tasks, but also preventing the presence of human error and even offering possibilities that would be unthinkable a few years back.

The need for automated systems has also been present in the biological fields of study. In the recent years, there has been an increasing interest in performing biological experiments using microscopes with live imaging techniques. When carrying out a study like these, there is a considerable amount of data that results from it. The analysis of all this information means a very complex and long examination that researchers have to perform. Therefore, in this case the introduction of automated systems can suppose a huge progress and relief for the biological research community.

More specifically, recently there have been several attempts to create automate systems that are able to perform the study of mitochondrial trafficking (the importance of these analysis will be explained in the following section). The main idea behind this kind of analysis is to track mitochondrial organelles along a neural axis on a sequence of images over time. A simple visual example has been included in Figure 1.1.

This type of research is a very time consuming task that is very often prone to human error. The images that scientists have to analyze are usually overpopulated, which usually causes moving elements to overlap with each other and therefore appear as hidden for a certain period of time. Moreover, there's usually an important amount of background noise present that can be wrongfully identified as organelles. Another difficulty presented by these images is that because they are usually captured with fluorescence and confocal microscopy (these techniques will be explained in greater detail in the following sections), bodies can go out of focus and therefore be difficult to identify in the image. Other features that make these images so complex are, to name a few, the mitochondrial jerky motion, the internal organelle's vibration and the temporal resolution of the imaging technique. Summarizing, when researchers carry out this type of studies they have to dedicate long periods of time analyzing these images.

Because of all these reasons, in the last 20 years there have been several attempts to automate this type on analysis. However, also because of all the reasons mentioned, this mechanization hasn't resulted as easy as expected. Many attempts have been carried out, however still nowadays there isn't a system that is able to accurately study all types of mitochondrial trafficking images.

This project aims to develop an automated system that researchers can rely on when performing mitochondrial trafficking analysis on neural cultures.

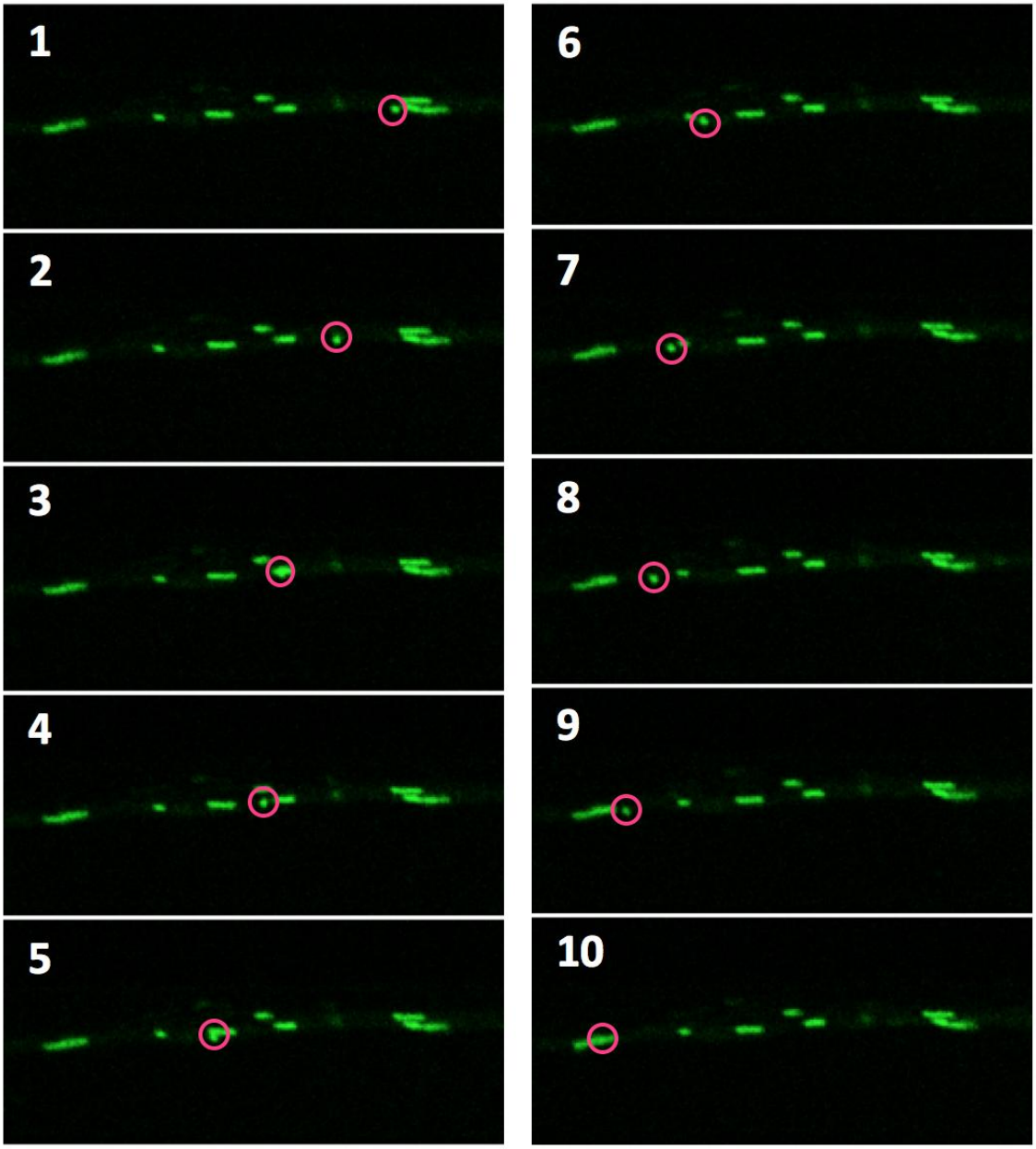


Figure 1.1. Representative example of sequence of images containing mitochondrial movement (zoomed area)

1.1. Mitochondrial trafficking

Mitochondria are known for being unusual organelles found in cells. They have an approximate size of 0.7 – 3 μm [1] and present their own distinct DNA and are surrounded by two membranes. They are often referred

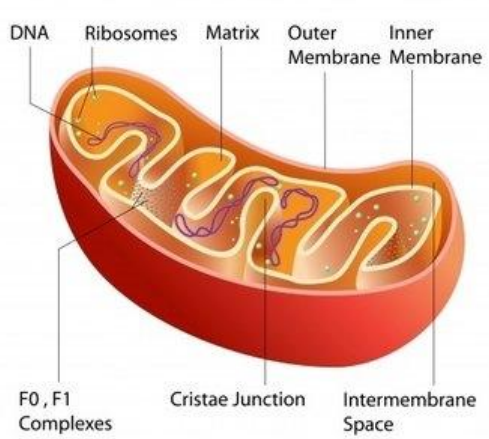


Figure 1.2. Structure of a mitochondria [2].

as the powerhouse of a cell, because their main function is to perform cellular respiration, creating ATP (energy) for the cell. However, that is not their only function, since they synthesize several compounds and steroids and are also a reservoir for calcium ions. All these features make mitochondria a key regulator of cell proliferation and death.

Neurons, which are the cells that make the nervous system, can present very long and complex axons (see Figure 1.3) with special energy demands. To sustain these structures, it is necessary that newly assembled mitochondria in the soma are moved to the axon and axon terminals and that damaged mitochondria (with defects in proteins or DNA) are reassembled or removed from the cell [3]. For this reason, in order to maintain energy balance and an adequate activity level, neurons need to present a correct distribution of mitochondria.

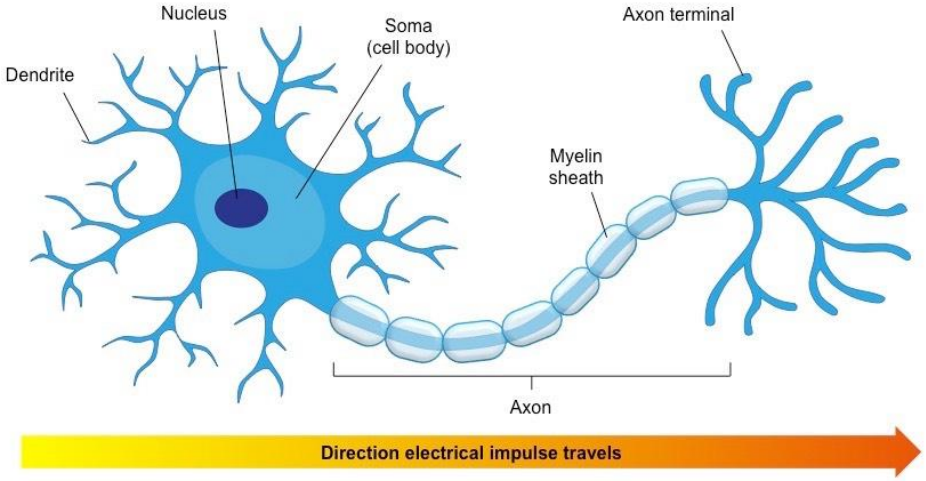


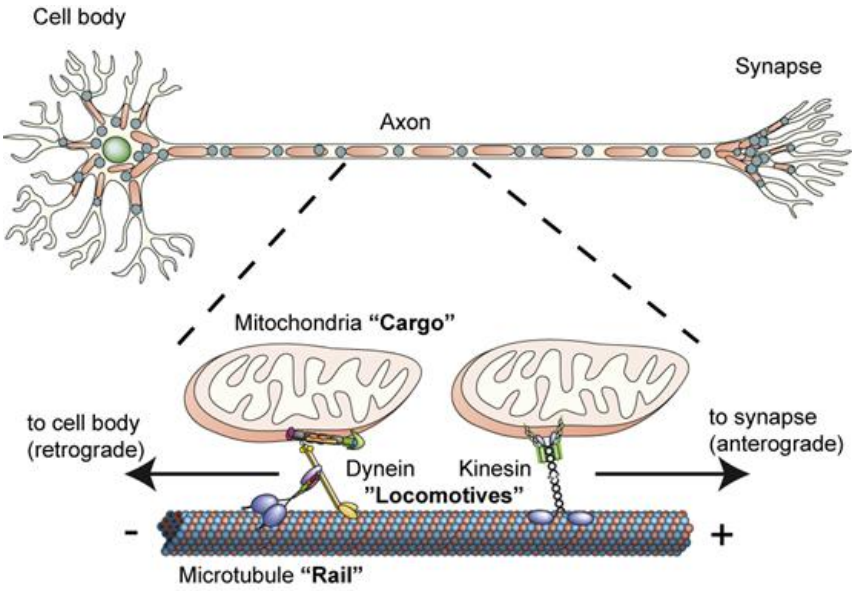
Figure 1.3. Structure of a neuron [4].

From all of this, it can be deduced that the movement and organization of mitochondria inside a neuron can determine the functionality of that cell. On the contrary, misregulation of the mitochondria's motility can lead to severe neural problems: neuronal dysfunction and degeneration.

A brief simplification of how the mitochondria moves along the neural axon is to consider that these organelles travel along highways that go from the body of the neuron to the axon terminals. These highways are in fact microtubules where mitochondria get attached to and move along them. Each axon can have multiple microtubules that can be tangled-up and in some cases mitochondria can jump from one microtubule to another [5]. All this internal structure causes the trafficking of mitochondria to be more complex than it initially seemed.

Moreover, mitochondria can travel in two different directions, anterograde and retrograde. Anterograde is the movement where mitochondria go from the soma to the synapse (axon terminals). On the other hand, the retrograde direction includes organelles moving in the opposite

direction, from the synapse to the soma. The meaning associated to these two directions is very different. Mitochondria moving in the anterograde are those that are healthy and well-functioning. On the contrary, organelles that have been damaged or present some type of defect present a retrograde movement in order to be transported to the cell body for destruction [6].



De Vos, Kurt J., et al. "Role of Axonal Transport in Neurodegenerative Diseases." *Annual Review of Neuroscience* 31 (2008).

Figure 1.4. Mitochondrial movement and its direction [7].

All these transport mechanisms play an important role in the functionality of a neuron. As stated previously, a defect or malfunction in these organelles or its capacity to move is crucial for the cell's health. This is relevant to such extent that mitochondrial trafficking abnormalities are involved in many neurodegenerative diseases such as Alzheimer's, Parkinson's, Huntington's, schizophrenia or Charcot-Marie-Tooth (CMT).

For this reason, nowadays a lot of research focuses on the study of these organelles and the analysis of their trajectories. There's a general interest in quantifying the movement of the organelles and finding the root problem that causes all the distortions in the mitochondrial trafficking, and consequently affects the neuron's functioning.

1.2. Live imaging techniques

Mitochondria are very small organelles that cannot be seen with the naked eye. Therefore, in order to get information about these bodies it is necessary to use microscopy techniques.

Microscopes were created in order to view objects and structures that couldn't be captured by the human eye before. Over the years, different branches of microscopy have appeared, but they can be grouped into three main categories: optical, electron and scanning probe microscopy.

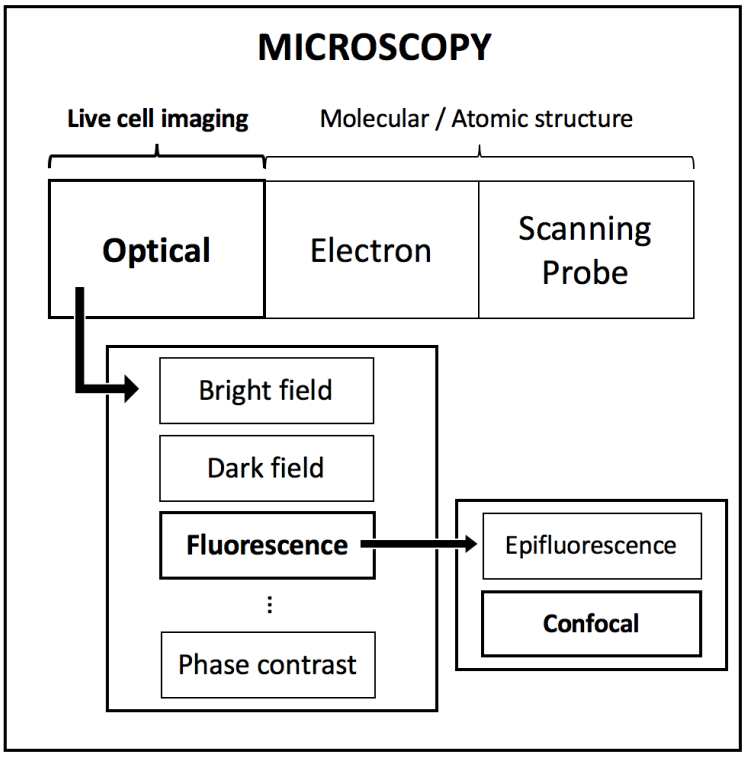


Figure 1.5. Schematic of the different microscopy techniques, in bold the ones used in this project

Optical microscopy, usually referred as light microscopy, first appeared in the 17th century and is the oldest microscopy technique. This method uses visible light and a single lens or a combination of them in order to enlarge objects. The visible light is transmitted through or reflected on the sample that is under study, which has to be placed on the focal plane of the lens. Throughout the centuries, many specific optical techniques appeared, like for example bright or dark field, phase contrast or fluorescence.

Electron and scanning probe microscopy are both sub-diffraction techniques, which mean that solve the problem of the diffraction limit resolution presented by optical microscopy [8]. The latter technique can only distinguish elements that are separated by 0.2 μm or more [9]. Electron and scanning probe microscopes end with this limitation and therefore increase their resolution by using an electron beam with a smaller wavelength than visible light and by including physical contact of the sample with the probe tip, respectively. Because of their high resolution, both techniques are usually used to study molecular and atomic structures. However, neither of them can be used to obtain live cell images because their samples need to be fixed (non-living) [10].

Consequently, the only available technique for live cell imaging is optical microscopy. However, as it was previously mentioned, there are many different optical methods. One of the most popular techniques is fluorescence microscopy.

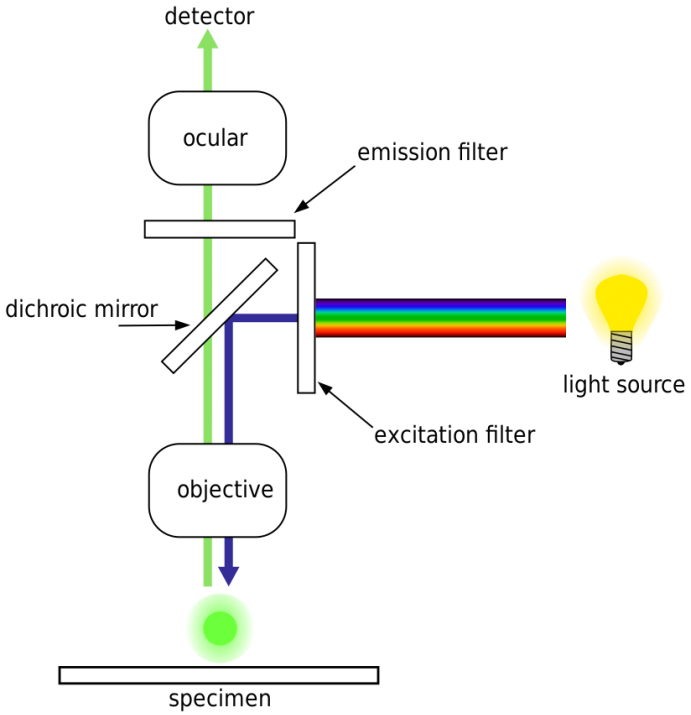


Figure 1.6. Schematic representation of the functionality of the traditional fluorescence microscope [11].

Fluorescence microscopy is a technique that uses fluorescence or phosphorescence to create images. More specifically, the fluorescence is achieved via fluorophores. Fluorophores are chemical compounds that can bond covalently with macromolecules and therefore be used as biomarkers. These compounds are able to absorb light energy of a specific wavelength and then emit light energy but with a different particular wavelength [12]. Each fluorophore absorbs and emits a determined light energy wavelength, thus obtaining different resulting colors: red, green, yellow, blue… The most common fluorophore is the Green fluorescent protein (GFP). As its name indicates, it emits green light when exposed to light from the blue to the ultraviolet range. The scientists responsible for the discovery and development of this specific fluorophore were awarded the Nobile Prize in Chemestry in 2008.

Summarizing, fluorescence microscopy is based on emitting a specific wavelength that will illuminate the fluorophores that will then emit a different wavelength that will be captured and responsible for the final resulting image. This type of microscopy is used for the study of structures and functionality.

This type of microscopy can be used on two different microscopes: epifluorescence and confocal. The main difference between both techniques, apart from the fact that the second uses a much more complex system than the first one, is that epifluorescence projects the light beam over a large

portion of the sample and as deep as it can, while the confocal technique only does it on a very small part and on a narrow depth level [13]. The confocal technique is able to reach a higher optical resolution than the traditional methods.

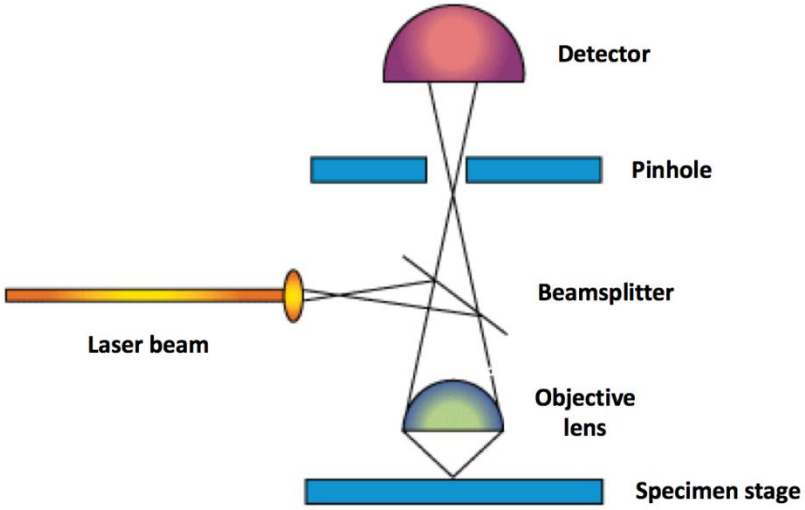


Figure 1.7. Schematic of the confocal microscope [14].

2. State of the art and previous work

2.1. Existing systems

Due to the importance of the meaning and consequences associated to the mitochondrial trafficking, scientist have been studying it for many years now. To carry this type of studies, researchers use live imaging techniques, as the ones explained in the previous section, to obtain images of how the mitochondria move along the neural axis.

In this kind of experiments there are many resulting images that have to be analyzed in order to be able to quantify the mitochondrial trafficking correctly. That's because in biological experiments you have to analyze n different samples to obtain reliable results and conclusions for a certain period of time, the longer the time of measure, the more images there will be to analyze. Moreover, usually these observations are made as a comparison between two or more types of specimen, therefore the number of images is doubled or even tripled. In conclusion, there is significant amount of images that have to be analyzed to carry out this type of research.

Originally, scientists used to take these images, lock themselves up in a dark room (to see the fluorescence images better) for long periods of time and check them one by one, looking for movement among a crowded lump of small objects and track it over different frames in time. Moreover, there can be overlapping between the organelles, reduction of their fluorescence intensity due to a change of plane, etc. All these circumstances cause the analysis to be very complex, time-consuming and prone to human errors.

As digital image processing techniques became more common in the 2000's, the first approaches to automatize this type of experiments were attempted. Since then, there have been many different tries and different methods applied. There are mainly two different types of approaches: those that use the kymograph created from all photograms [15], [16] and [17] and on the other hand there are those that use motion tracking, applying segmentation and tracking techniques in each photogram in time [18], [19] and [20]. Graphical examples of the steps that follow each technique can be seen in Figure 2.1 and Figure 2.2.

Many attempts have been done and some of them might detect and quantify mitochondrial movement more accurately than others. However, none of the methods has been proven to be good enough to completely avoid human intervention and therefore perform a fully automated analysis on its own on any kind of mitochondrial motion image. Although software are more complex every time,

an algorithm that offers reliable results that allow to withdraw clear conclusions has not been created yet.

One of the challenges that these types of automated programs face is the variability between experiments. Fluorescence intensity, SNR ratio, axonal distribution, to say a few, can vary a lot from one research center to another or even from one researcher from another, even in the same center. Therefore, it is difficult to create a code that is able to extract information from such a large diversity of images. Usually, the best way to deal with problem is either to have some inputs parameters in the code that can be readjusted for each experiment depending on its characteristics or to have a very protocolized procedure to capture the images, in order to avoid the variability between trials.

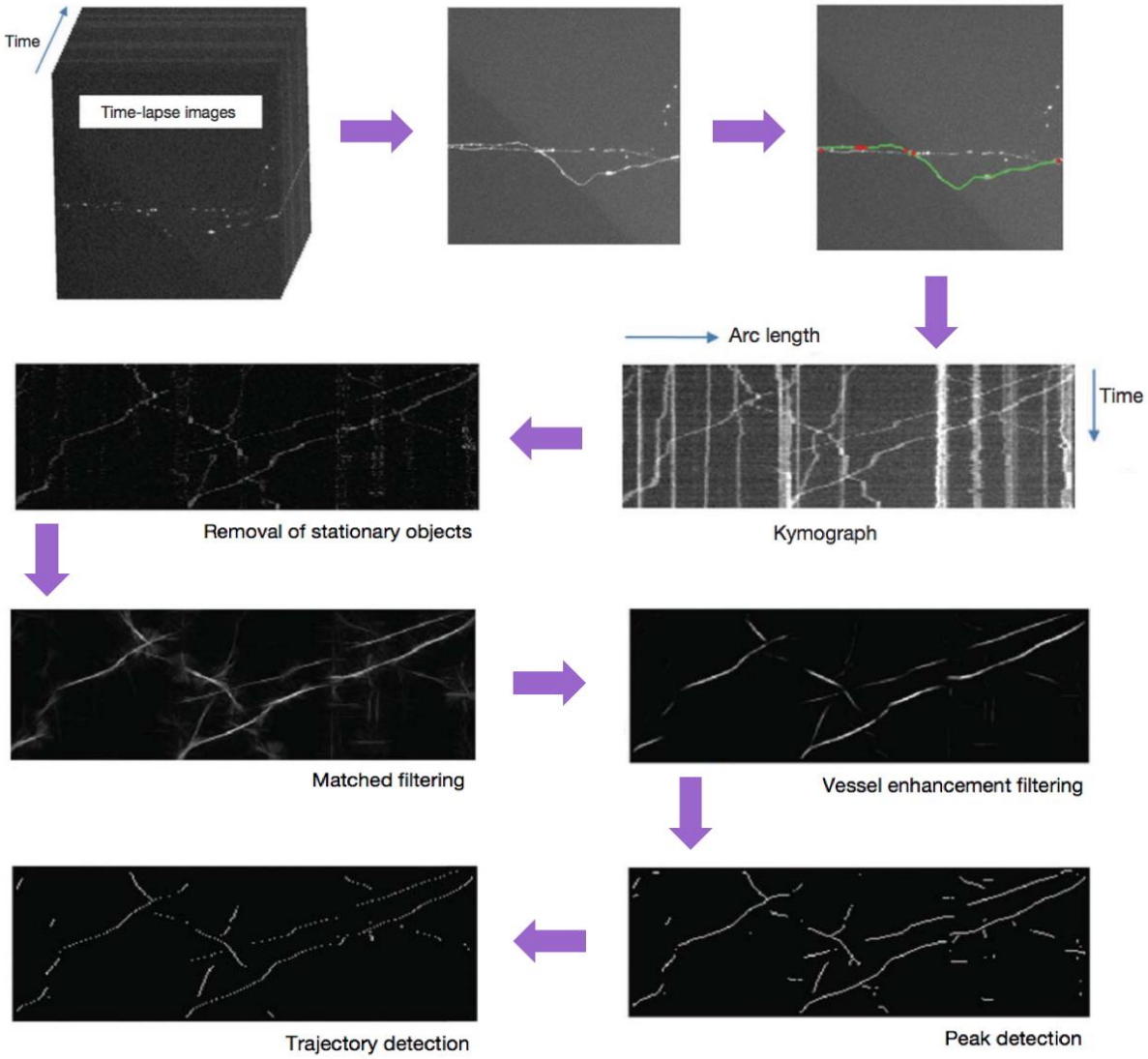


Figure 2.1. Basic pipeline of a kymograph approach for mitochondrial tracking [15].

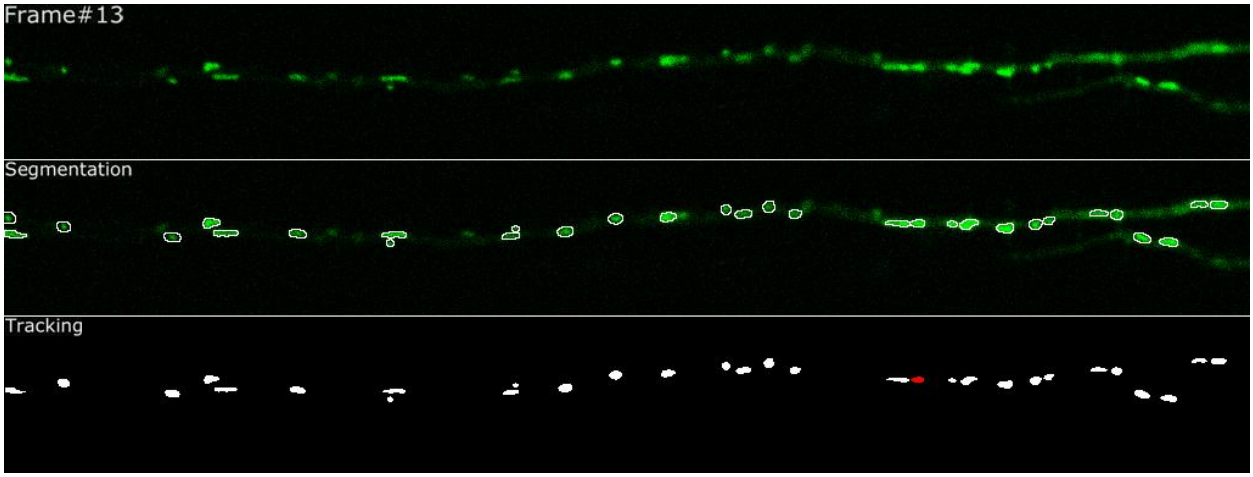


Figure 2.2. Basic steps of the motion tracking pipeline.

2.2. Project's framework

There's a laboratory in Barcelona's Children's Hospital Sant Joan de Déu dedicated to Neurogenetics and Molecular Medicine. They focus on the study of neurodegenerative rare diseases, like for example Charcot-Marie-Tooth (CMT). This disorder is associated to the deficit of the GDAP1 protein. In their line of research, they are looking into an existing relationship between the deficit of this protein and the mitochondrial motility. As previously stated, mitochondrial activity affects neural functionality. This study is carried in order to further understand the disease and therefore be able to treat it or even better, cure it.

One of their research phases included a breed of two different specimen of mice: wild-type (WT), which did not present any type of genetic mutation, and knockout (KO), which presented genetic mutations so that there was no coding for the GDAP1 protein. Afterwards, they wanted to analyze the differences between both specimens in order to see the effects caused by the lack of GDAP1. To do so, they wanted to quantify the mitochondrial movement in both cases and see the differences between them.

After performing the experiment and analyzing the resulting images one by one, they obtained the final results of the analysis (further on this information will be referred as the Ground Truth). However, they realized that it was a very high time-consuming task and very prone to human error. Moreover, they wanted to protocolize and improve the technique in order to be able to repeat in the future.

From this need was born the collaboration between the Neurogenetics and Molecular Medicine laboratory and the LASSIE Laboratory in EEBE, which focuses on the research of image and signal processing. The collaboration consisted on the creation of an automated program by the UPC

research group, that could perform an automated quantification analysis of the mitochondrial trafficking.

Initially, a PhD student from the LASSIE Lab started working on it. It was initially decided to work with a program based on a motion tracking approach. The reason behind this choice is that the images provided by the biology team contained highly dense dendrite structures. Moreover, the images presented a very low frame rate and a low SNR. This combination of features made them very difficult to analyze via kymographs. As seen in Figure 2.3, it is almost impossible to see any clear trajectories described by organelles in this experiment, because there's a lot of overlapping and chopped trajectories.

Figure 2.3. Representative example of a kymograph from one of the experiments that had to be analyzed.

A first approach using motion tracking was programmed and initial results were obtained. However, these first results did not look anything like the ones that had been previously found by the Sant Joan de Déu's laboratory (considered as ground truth, GT data). The automated program wasn't able to detect almost any type of mitochondrial movement.

Afterwards, a few modifications were included into the code and the results offered by the program were completely different. At this point, the code was detecting too much movement in comparison to the GT data. Approximately, the number of mitochondria moving compared to the number found by the biologist was twice as much. So, again, there were still modifications left to do.

Here at this point is where this project starts. The PhD student had to leave the project and I was offered to continue the project as my final thesis.

This project is carried out under a collaborative scholarship awarded by *Ministerio de Educación, Cultura y Deporte*. Thanks to this scholarship I was able to develop this project in two different locations, LASSIE Lab in EEBE and the Neurogenetics and Molecular Medicine laboratory in Fundació Sant Joan de Déu. Being able to work very closely with biologists and engineers at the same time allowed me to have a deeper understanding of the two fields of work involved in this project without underestimating any of the two disciplines.

3. Project Goals

The goals of this project can vary depending from the point of view considered. It has to be noted the difference between the goal of the initial collaborative project and the goal of this final thesis.

The main goal of the collaboration between the hospital and the university is to create a **reliable and robust program that can track and quantify mitochondrial movement automatically**, without any user intervention. The purpose of the creation of this program is to avoid manual time-consuming analysis and prevent human errors due to each person's subjectivity.

Because this thesis was started once an initial code was already created, the main goal of this project is to **modify and improve the existing code** in order to be able to finally create a code that offers reliable results and can be trusted to work automatically without human supervision. However, this main objective involves lots of smaller steps and goals that need to be achieved in order to fulfill the final intention.

The first step is to really **understand the meaning and background of the biological problem** that is being considered in this project. The motion of mitochondria along a neural axis has many meanings associated and it is difficult to interpret these images without this knowledge.

Moreover, in order to be able to modify the code it is necessary to **understand the original algorithm** that exists up until this point. The initial code already contains many lines of code, and it requires a meticulous analysis of each command in order to later introduce modifications to improve it.

Secondly, once the code has been understood, it is necessary to **find the flaws** that cause the program to not work correctly. This requires many hours of code analysis to really identify which parts of the code are causing the errors.

Moreover, when the weaknesses have been recognized, they have to be modified. This steps involves two types of work: **modify code that already exists** and for some reason isn't working properly and on the other hand, **include new actions inside the code** to perform new computations that had not been included in the previous version of the program. These modifications and additions can be done in any part of the algorithm: noise filtration, segmentation, tracking, post-tracking analysis…

Then, it is required to **validate the results** that the code offers after all modifications. In other words, it has to be checked that the results that are offered by the program match closely the results that were found during the first analysis performed by researches in Sant Joan de Déu (we refer to it as the GT). If the program's results aren't close enough to the ground truth, then the previous stage is repeated again (more modifications and new additions are included) until the results are admissible.

Until here, all the steps necessary to achieve the main goal of my project have been described. However, we decided to add one extra phase to really finalize the project: **validate our software against an already existing one**. This step is useful to fully analyze the extent and impact of our project in the research society and the possibilities that can be associated to it.

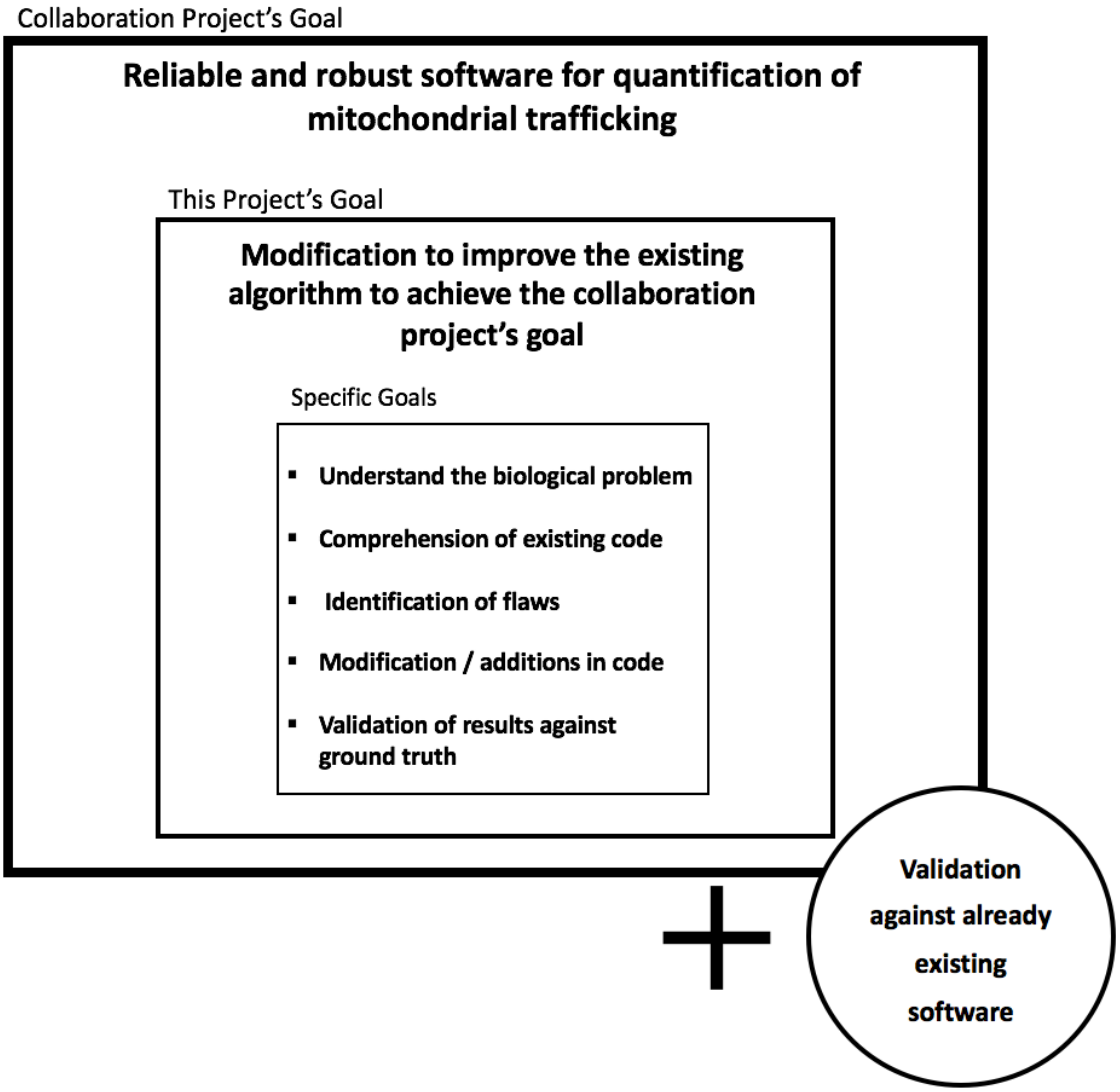


Figure 3.1. Schematic representation of the goals of this project, placed inside the framework of the collaboration's goals

4. Materials & Methods

4.1. Materials

This section includes the explanation of how the team at the Neurogenetics and molecular medicine laboratory obtained the images that were used to carry out the quantification analysis of mitochondrial motion. It describes the procedure used to prepare and cultivate the cells in order to be able to obtain images that contain mitochondrial movement along a neural axis. All this information has been obtained from [21].

4.1.1. Experimental data

A total of 71 experiments were recorded at a sampling period of 5.27 s an image size of 1024 x 128 pixels, a physical pixel size of 0.12 μm and 16-bit depth. 14 experiments were recorded during a period of 5 minutes (57 frames) and 57 experiments during 131 seconds (25 frames).

4.1.2. Image acquisition

In-vivo mitochondrial movement in neurons was experimentally studied using microfluidic chambers. The microfluidic device is composed of two open culture chambers connected by a parallel array of microchannels. The system allows fluidic separation of axons from the soma and permits observing mitochondrial movement along oriented axons. Axonal microchannels were labeled with the mitochondria-specific dye (Mitotracker Green) and recorded in a climate controlled chamber (In Vivo Scientific) at 37 ºC and 5% CO_2 using conventional confocal microscopy (SP2, Leica). Fluorescent marker was stimulated using a 488 nm argon laser and the optical setup included a x63 oil immersion lens with x2 zoom and a pinhole of 1 AU.

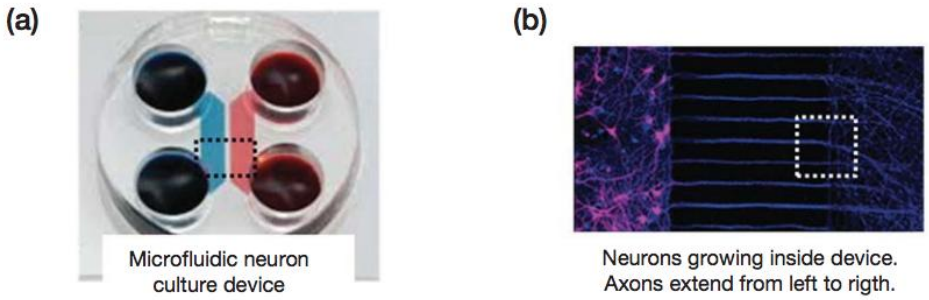


Figure 4.1. Microfluidic chamber and neuron growth in them [15].

4.1.3. Embryonic motor neuron primary culture

Embryonic motor neuron (MN) cultures were prepared from 13.5 embryonic day (E13.5) mouse spinal cord. Briefly, mouse embryo spinal cords were dissected and the dorsal half removed. Ventral spinal cords were dissociated mechanically after trypsin treatment (0.025% trypsin in HBSS), and collected afterwards under a 4% bovine serum albumin cushion. The largest cells were isolated by centrifugation (10 min at 520 g) using iodixanol density gradient purification. The collected cells were finally suspended in a tube containing MN complete medium: Neurobasal (Life technologies) supplemented with B27 (Life technologies), 2% horse serum, 1x glutamax (Life technologies), and a cocktail of recombinant neurotrofins: 1ng/mL BDNF; 10ng/mL GDNF, 10ng/mL CNTF, and 10ng/mL HGF (PreProtech). 10 μM AraC (Sigma-Aldrich) was added to the culture medium to limit the growth of non-neuronal cells. MNs ($3x10^5$ cells) were added to the proximal chamber of prepared microfluidic devices and grown in a 5% CO_2 incubator at 37 ºC.

4.1.4. Preparation of Microfluidic chamber

Microfluidic devices with 450 nm microgroves (Xona Microfluidic, Catalog SND450) were cleaned of surface particles using adhesive tape and sterilized in 70% ethanol for 2-3 hours. Devices were dried completely under sterile conditions, attached to treated Poli-L-ornithine sterile 22 mm^2 coverslips (Thermoscientific) using gentle pressure from blunt sterile forceps and placed in 6 well culture. Each device was designated a proximal and distal side. Chamber were filled with neuron medium and equilibrated for 2 hours. The medium was removed and Laminin was added and incubated overnight at 37 ºC. Laminin was removed prior to addition of neurons.

4.2. Methods

In this section, the pipeline of the algorithm is described. The code has been created with the software MATLAB. A wide variety of different tools have been used: statistical characterization, thresholds, filtering or feature extraction. All the steps for the image processing pipeline are explained in detail in the following subsections.

As previously explained, this project is a continuation of an initial project that was already started. Therefore, the final algorithm includes sections of code that have only been edited by Alex Vallmitjana, while there are other lines of code that have been introduced as completely new. In conclusion, the pipeline that will be discussed in this section is a combination of work of two different persons. In each step, it will be explained what the original code included and what modifications or additions were done to it in order to improve it.

As initially mentioned, before being able to modify the code, it was necessary to understand it and recognize the flaws in it. After a deep analysis of the instructions in the code and the results offered by each of them, several weaknesses were identified. Each of these flaws can be included in one of the main sections of the code:

PREPROCESSING & SEGMENTATION	Overdetection of mitochondria during the segmentation process
CLUSTERING	Disadvantages associated to the clustering process
TRACKING & TRAJECTORY ANALYSIS	Incorrect or incomplete final trajectories
	Overdetection of mitochondrial movement (moving tracks)

Table 1. Weaknesses identified in the initial version of the automated system

After establishing all the weaknesses of the algorithm, modifications were included in each corresponding section to improve it. In the following section it will be described in detail the characteristics of the flaws, the problems linked to it and the solutions that were adopted in order to solve them.

Once all modifications were added, the final pipeline includes all the following steps:

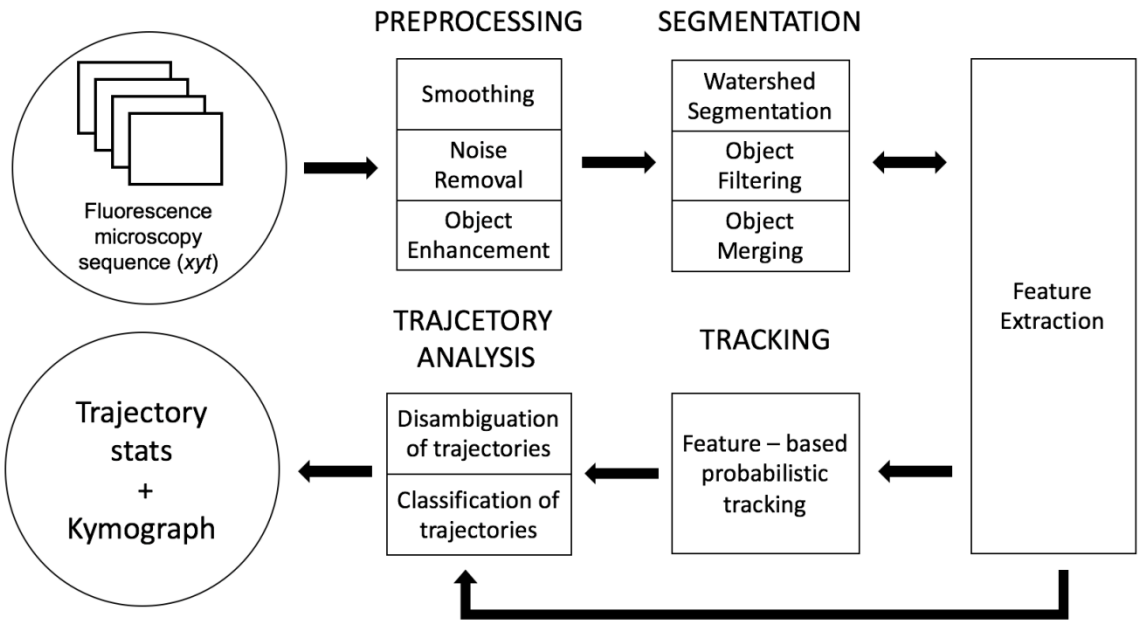


Figure 4.2. Diagram representation of the algorithm's pipeline

4.2.1. Image data preparation

First of all, it is necessary to transform the data that was originally created by researchers in Sant Joan de Déu into data that can be read and analyzed by MATLAB.

Initially, the experiments were recorded as videos in *.lei* format with the corresponding frame rate. Afterwards, the *.lei* videos were converted into *.tif* image sequences using the software ImageJ. Then, for each experiment, MATLAB read the corresponding image sequence and loaded all images from each sequence into a 3D volume (*xyt*). This final volume is the object that will be used when doing the following analysis.

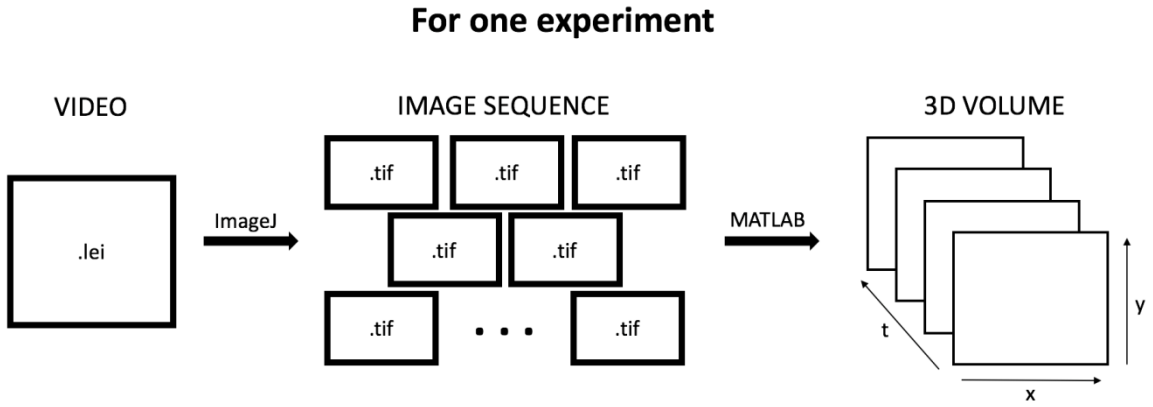


Figure 4.3. Schematic representation of the video to image data transformation process

4.2.2. Preprocessing

In most of the images there is a non-negligible amount of background noise that affects the segmentation of the individual mitochondria. Hence, a preprocessing step has been introduced in order to minimize as much as possible the residual fluorescence at the microtubules.

The first weakness that was identified was the overdetection of mitochondria during segmentation. However, it was mentioned that it involved not only the segmentation stage but also the preprocessing one. The reason that lies beneath this flaw is the presence of noise, that is confused with real mitochondria during the segmentation process. However, if we are able to remove this noise prior to the segmentation step, this problem can be significantly reduced.

4.2.2.1. Original code

The original code already included several steps that aimed to reduce the residual background fluorescence. First, convolution by a Gaussian filter with σ = 0.2 μm (a filter of 3x3x1 voxels) was applied, in order to obtain smooth surfaces.

Second, a thresholding baseline removal was applied by using a pixel intensity threshold T_I defined for each experiment as:

$$T_I = \mu_I + 1.5\,\sigma_I \quad (1)$$

where μ_I, σ_I are the mean and standard deviation of pixel intensity in the 3D volume, respectively. More precisely, μ_I is computed as the mean fluorescence intensity found when taking into account all pixels in all frames in time and σ_I is the standard deviation of this mean.

Finally, the threshold value is used so that all pixels that present an intensity level below T_I are removed by being set to zero, therefore converted into black pixels and are now part of the background.

4.2.2.2. New code

After taking a deep look into the code, it was clear that the two steps applied during the preprocessing stage were not enough to remove the large amount of background noise present in the images. The impact this noise had in the final results of segmentation was too high and needed to be controlled. For this reason, two modifications were added into the algorithm.

Both steps were studied, and it was concluded that although the second step (pixel intensity threshold) was working properly, the impact that had the Gaussian filter could be improved. To do so, two options were considered:

- Use a 3D Gaussian filter ✗
- Use a larger 2D Gaussian filter ✓

The 3D filter was designed with the idea of removing noise that appeared and disappeared from frame to frame. Therefore, if there was some noise present in one frame that disappeared in the next frame, the filter would mitigate the intensity of the noise. However, the main problem of this technique was that it didn't just affect the noise removal, but it also removed the real mitochondrial movement that was occurring. Therefore, this idea was rejected.

The next option was to increase the 2D Gaussian filter, because the noise particles were too large and the previous used filter was not large enough to filter them. The final filter used can be seen in Figure 4.4. Instead of using a 3x3 filter, it was increased to a 5x5 Gaussian filter, that was able to fade the noise better than previously.

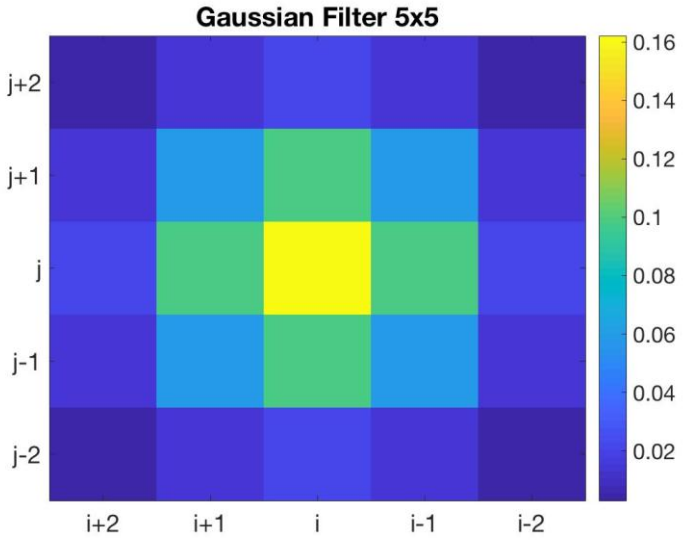


Figure 4.4. Gaussian filter 5x5 used in the final version of the algorithm

Regarding the same flaw that was previously mentioned, the overdetection of mitochondria, it was noted that the problem wasn't just noise that was identified as organelles. The second cause of the overdetection was that real detected mitochondria was segmented into more than one object. Therefore, one real mitochondria could be identified by the algorithm as two or more different organelles.

The first though was that during the threshold removal there were some pixels inside the organelle structures that were removed, creating *holes* inside the objects. Therefore, the first approach was to *fill* these *holes*. So, the last stage added was a flood-fill operation on background pixels using a filter to enhance horizontal structures, which is the usual shape of mitochondria. The size of the filter was limited, so that it wouldn't create a union between originally separated objects. The filter can be seen in Figure 4.5.

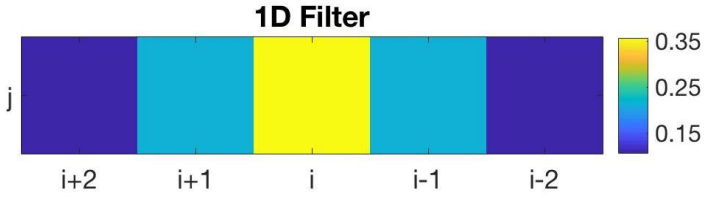


Figure 4.5. 1D filter used to boost horizontal structures (mitochondria)

This technique was not useful to solve all cases were one single mitochondrion was segmented into various objects, but it did solve it in a few situations. That's why another new step to solve this issue was included in the segmentation process as it will be seen in the following section.

The final result after the preprocessing steps is shown in Figure 4.6. The figure offers, from an example of a representative experiment, a comparison between an original frame of the experiment without any type of modification from the original version and the filtering obtained using the original code created and the results offered by the modifications added in the final version. As it can be seen, the filtration in the new code is more intense than previously. Also, in both filtration cases there's a significant difference with respect to the original frame. This figure also shows how although a lot of noise is removed, there's still a lot of noise left in some areas (see right corner of zoomed out image). This remaining noise will lead to complications during the segmentation step that will have to be solved later on.

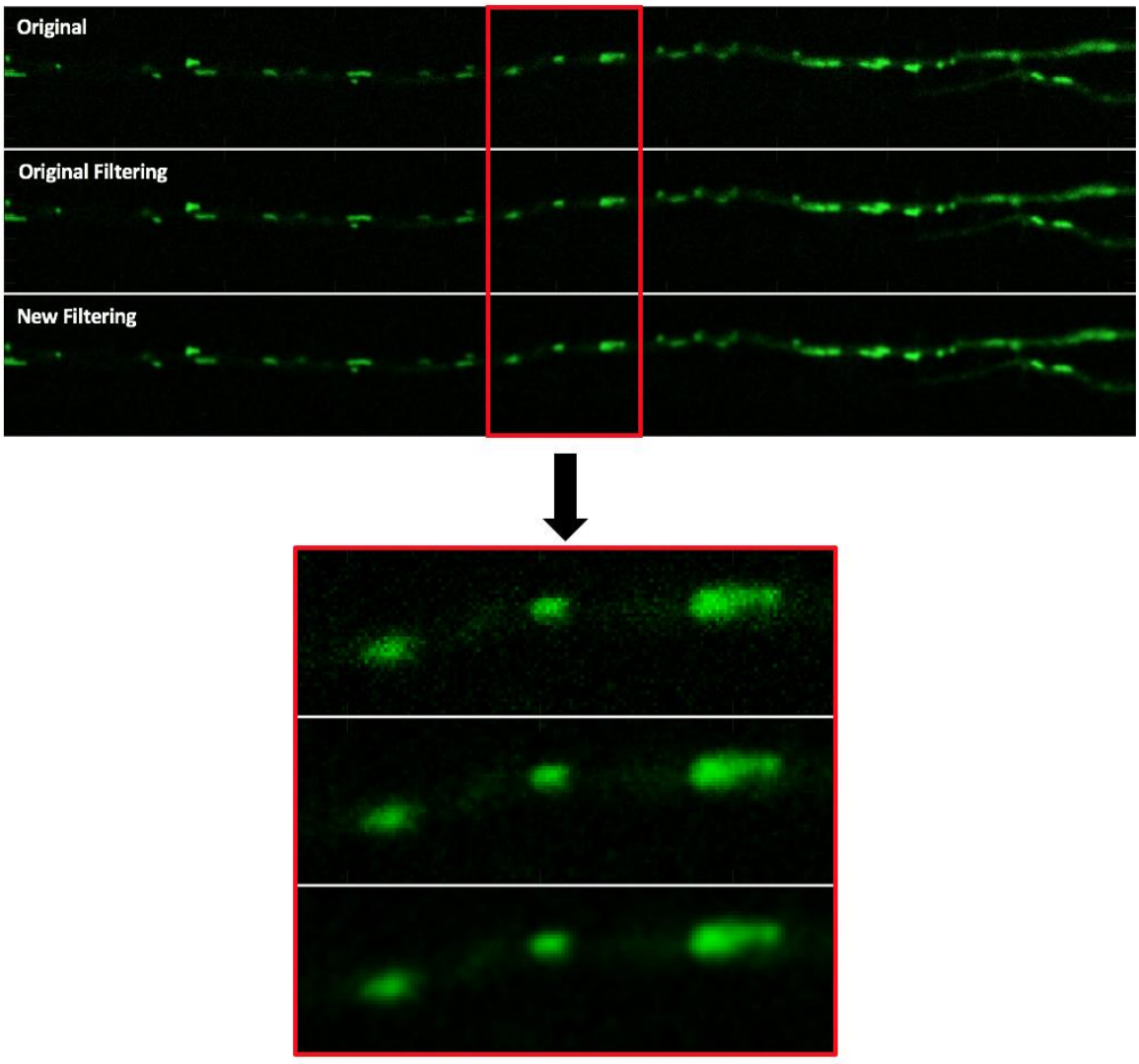


Figure 4.6. Representative comparison of the original frame with the results offered by the filtering during the preprocessing phase in the two codes (original and modified).

4.2.3. Segmentation

The main purpose of this phase is to recognize mitochondrial structures on each frame in time and correctly segment them, which is equivalent to detect exactly what pixels are part of a mitochondrion's structure and which ones aren't.

This step is crucial in order to obtain reliable final results. If the segmentation misses real objects, then the tracking process won't be able to match mitochondria in time correctly. On the opposite, if the segmentation identifies background noise as objects, it will mislead the tracking process afterwards. Therefore, improving the segmentation directly implies an improvement in the reliability of the final tracking results.

As previously stated, this step is the main responsible stage for the first weakness mentioned: overdetection of mitochondria. Although this problem was already treated in the preprocessing stage, the goal of the new code in this section is to completely solve the problem, so that the final segmentation offered by the program is reliable and doesn't confuse the tracking process.

4.2.3.1. Original code: Watershed Segmentation

In the original code, the segmentation stage was only made up of a watershed segmentation technique. It is a customized watershed method with a stop rule, specifically designed for this algorithm, and it is applied to each frame. This method resembles a topological surface, where the brightest pixels (usually objects) represent the deepest points and the darkest (background) represent the highest points of the surface. The technique can be thought as if the overall area were to be flooded and then lower the water level gradually. All isolated puddles with a minimum size of 1 μm^2 that are left are considered as candidates. Later, the water level is reduced even more in order to subdivide the regions as long as the subregion has a minimum depth of 0.2 (normalized intensity). A visual representation is included in Figure 4.7. All these parameters mentioned have been chosen heuristically and can be tuned according to the size of the mitochondria found in images.

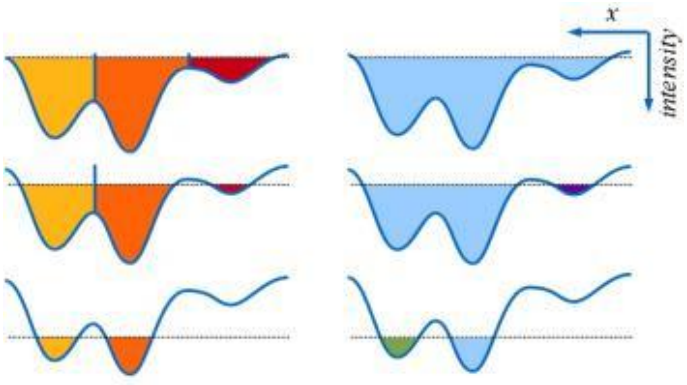


Figure 4.7. Graphic representation of the Watershed segmentation fundamental idea [22].

After the watershed segmentation, the original code includes a step of feature extraction. In this case, it involves the computation of the weighted centroid (x and y coordinates) and the area of each segmented element. The weighted centroid of a body is the central region of it, taking into account not only its shape and location but also its intensity distribution. This feature is computed using the *regionprops* command in MATLAB. On the other hand, to compute the areas of all organelles, a customized program was used, since it was considered that this costume created function computed the area in a more approximated way than the *regionprops* command. In the initial code, all these features are only later used in the feature based tracking process.

4.2.3.2. New code: Object merging and filtering

After taking a look at the results offered by the watershed technique, it was obvious that some elements weren't correctly segmented. In general, there were two typical mistakes:

- Segmentation of one mitochondrion into multiple elements
- Segmentation of background noise as mitochondrial structures

Modifications were needed in order to solve these two flaws. Altering the watershed segmentation process was not an option, since the flooding process is unable to distinguish between noise and mitochondria. Therefore, it was decided to go for a few a posteriori fix-up solutions, that apply to the previously mentioned flaws respectively:

- Object merging: to join oversegmented mitochondrion
- Object filtering: to remove segmented noise

Object merging

The object merging process is applied right after the segmentation of each frame. This procedure consists on considering all the labelled elements that are in contact and determine whether they are part of the same mitochondrion or not. If an original mitochondrion has been incorrectly divided into multiple organelles, the two elements need to be merged together. However, if the elements belong to two different real mitochondria, then it means that the segmentation process worked properly and no further actions need to be taken.

To perform these tasks, the program has a loop that analyzes all tags that are in direct contact with other tags. For each of these cases, the boundary region between the adjacent tags is examined and compared to the bodies' regions.

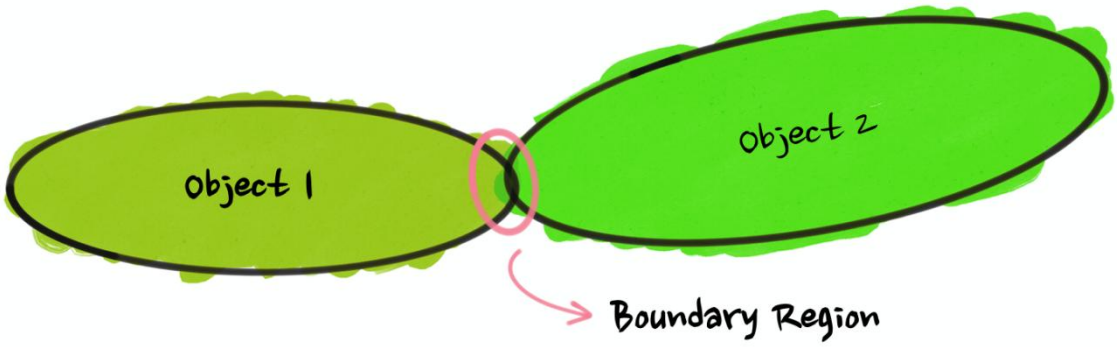


Figure 4.8. Schematic representation of two adjacent mitochondria and the boundary region between them

The tricky part here was the lines of code necessary to find the boundary region between the adjacent tags. The issue was that there were many different cases that involved many different geometries. For example, one boundary region could be shared by three or more elements, particles as thin as a 1 pixel wide could be involved in the analysis, etc. Three versions of code were done in order to finally achieve a successful result that could be used for any case to identify the boundary region between adjacent bodies.

The first attempt included a series of steps that have been represented in Figure 4.9. We can see that there are two cases of two adjacent labelled tags in the originally labelled image. The goal of the program is to find the boundary regions between these tags to later study its intensity in relation to the tags' intensities. So, to find the boundary regions, the labelled image was used to obtain a mask that would be of 0's and 1's, merging together adjacent structures, and also a gradient image, that contained only the perimeters of the tags. Then, an erosion was performed on the original mask and then a convolution between the gradient image and the eroded masked would result in a final image that only contained the boundary regions between adjacent tags. However, this technique was not always successful when dealing with very thin elements, because the convolution between the eroded mask and the gradient image would not leave any boundary regions behind. Therefore, a second attempt was performed.

In the second attempt, only the labelled and gradient image were used. Instead of using masks, the boundary regions where found using the gradient image and looking at what gradients existed between different tag labels. However, this version wasn't always useful either, since in some specific cases the gradient between multiple labelled tags could not be found present in all of them.

Finally, one last successful attempt was done. However, this last step didn't include any masks or gradient images. Instead, in the labelled image it was checked tag by tag if any of the tag's pixels were in contact with another labelled body. If so, that pixel and the pixel in contact would be part of the boundary region. Although this is much computationally expensive than the previous versions, it was the only technique that offered the desired results, so no further modifications were included.

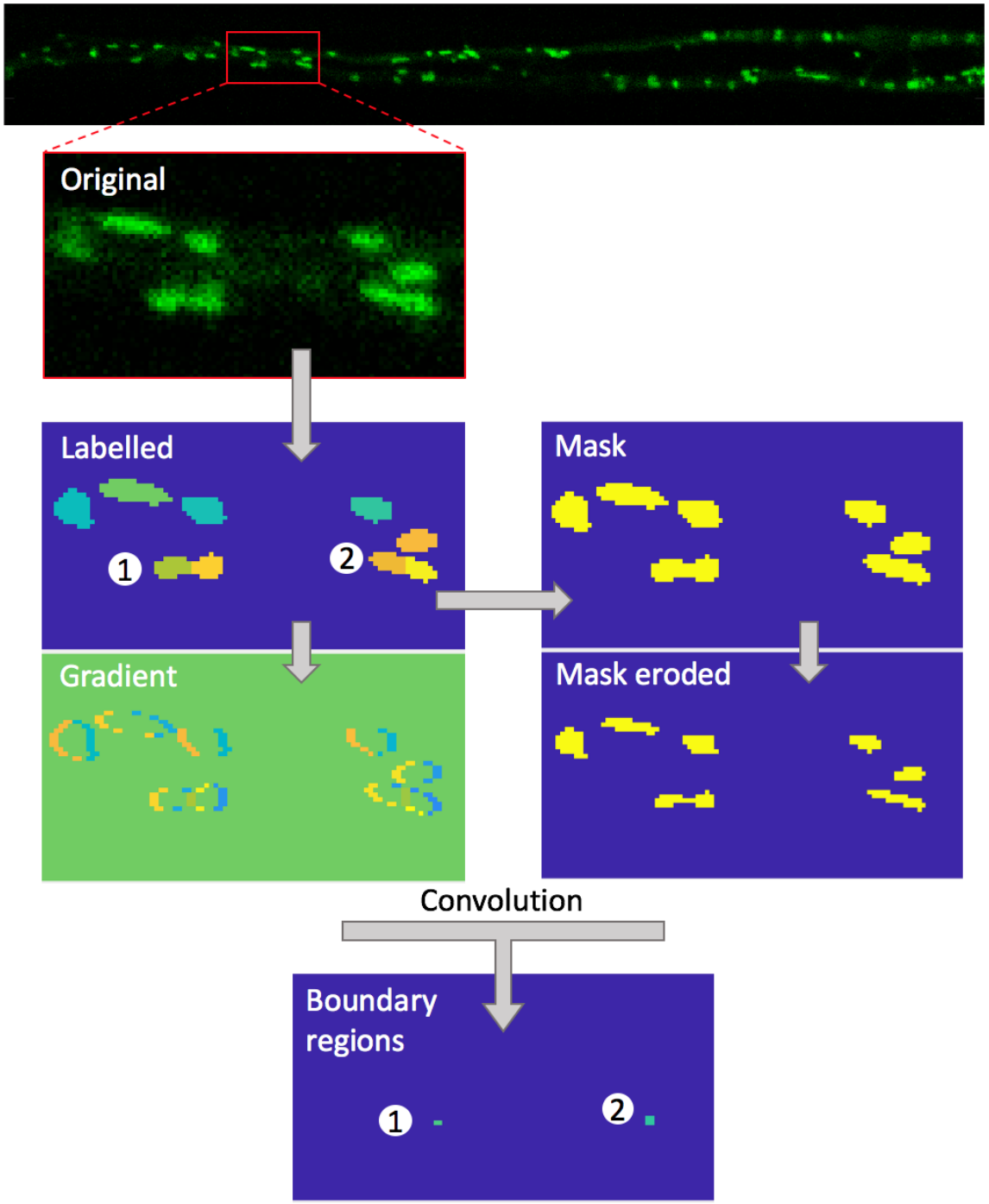


Figure 4.9. Schematic representation of the first attempt to an algorithm that can find the boundary regions of adjacent tags

Once the boundary region is successfully found, its fluorescence has to be compared to the adjacent bodies fluorescence to determine if they have to be kept as separate tags or it's necessary to merge them together.

If the fluorescence intensity in the boundary region is much lower than the mean intensity found in both bodies, then it is clear that the bodies are in fact two separate bodies. However, if the difference

of intensity isn't noticeable (smaller than a certain proportion) it is considered that the tags have been mistakenly divided during the segmentation process and need to be merged together.

To put these words in a much simpler expression, two contiguous labelled regions will be kept as two independent bodies if the following condition is met:

$$\overline{FI}_{BR} \leq k \cdot \frac{\overline{FI}_{O1} + \overline{FI}_{O2}}{2} \quad (2)$$

Where $\overline{FI}_{BR}$ is the mean fluorescence intensity of the pixels found at the boundary region and $\overline{FI}_{O1}$ and $\overline{FI}_{O2}$ are the mean fluorescence intensity of the pixels found at each of the adjacent objects being analyzed. Finally, k is the parameter that determines the proportion that relates both intensities (at the boundary and at the bodies). If $k = 1$, then the formula establishes that the two objects will be kept as two separate objects as long as the mean intensity at the boundary is lower than the mean intensity in the objects. We decided to use $k = 0.7$, meaning that it is allowed for the boundary to have a little less fluorescence intensity than the bodies. The value of k was heuristically determined, by the examination of different experimental conditions. It has to be kept in mind that these images were acquired by confocal techniques and therefore if part of a mitochondrion leaves the plane of interest, its intensity decreases on the images, but the mitochondrion is still one single organelle.

To sum up, the two segmented areas will be kept as separate as long as the intensity at the boundary is much lower than the mean intensity found on both segmented areas. Otherwise, the two objects will be joined together and merged as a single element. A visual representation is offered in Figure 4.10, where both cases are exemplified. In one case, a few original mitochondria that have been subdivided into multiple objects during segmentation are merged together individually. On the other side, there're two examples of several mitochondria that are in direct contact and therefore are candidates of the merging process. However, because they are in fact separate bodies, the algorithm is able to see the separation between them and keep them as independent elements.

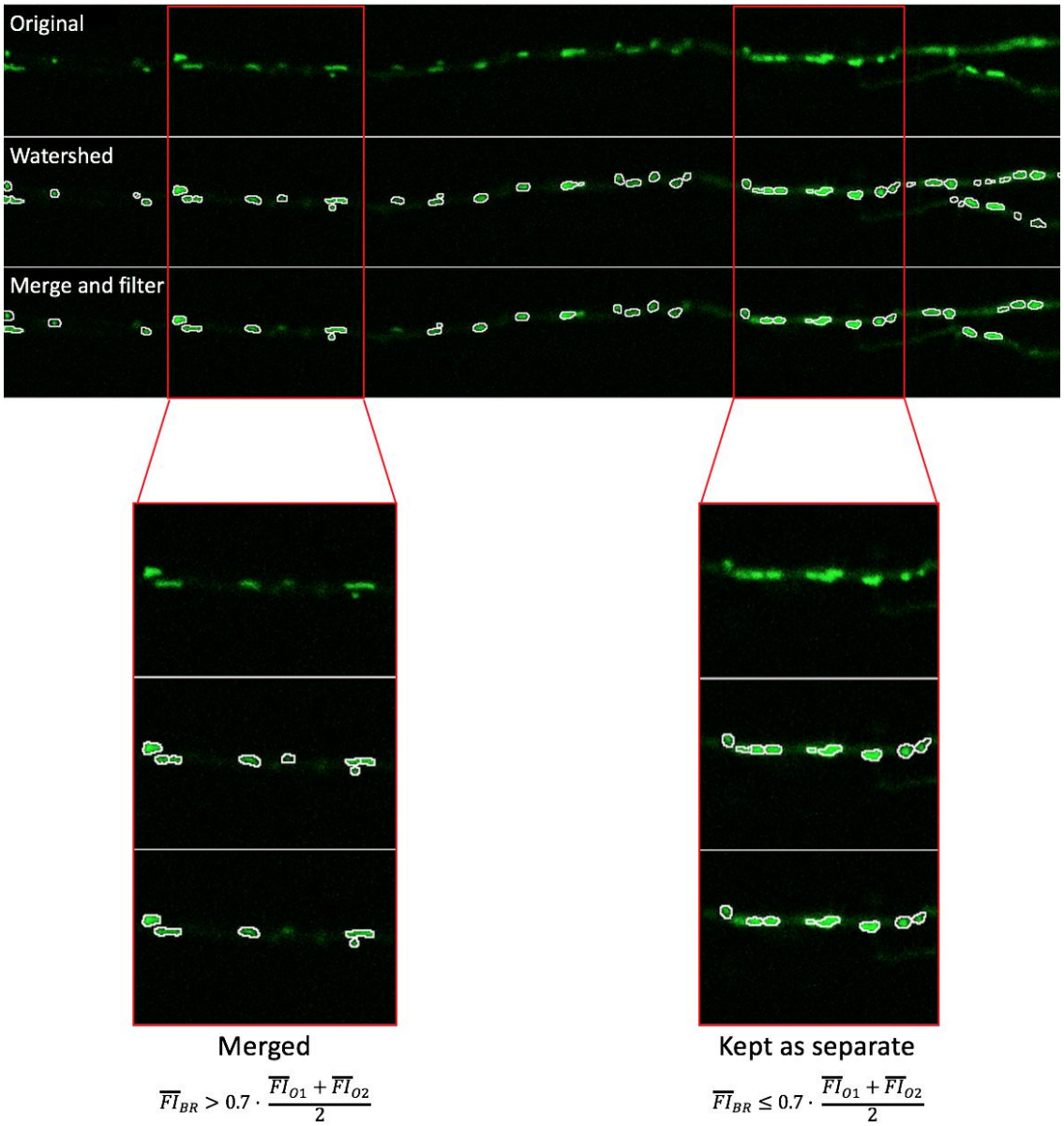


Figure 4.10. Representative example of the results offered by the watershed segmentation and merging techniques

Object filtering

As well as in the original code, the new version also includes a feature extraction section where each element is analyzed and certain characteristics are obtained. However, a few additions were done in comparison to the original algorithm. In the initial approach only two features were extracted: center of masses and area. The latest code includes a few additional features:

- Maximum pixel intensity: maximum value of pixel intensity found in the segmented region
- Major axis length: length (in pixels) of the major axis of the ellipse that has the same normalized second central moments as the region, returned as a scalar.

- Ratio between major and minor axis length: which relates the major axis length with the minor one. This measure is used to have an idea about the morphology of the object, whether it has a more elongated or rounded shape.

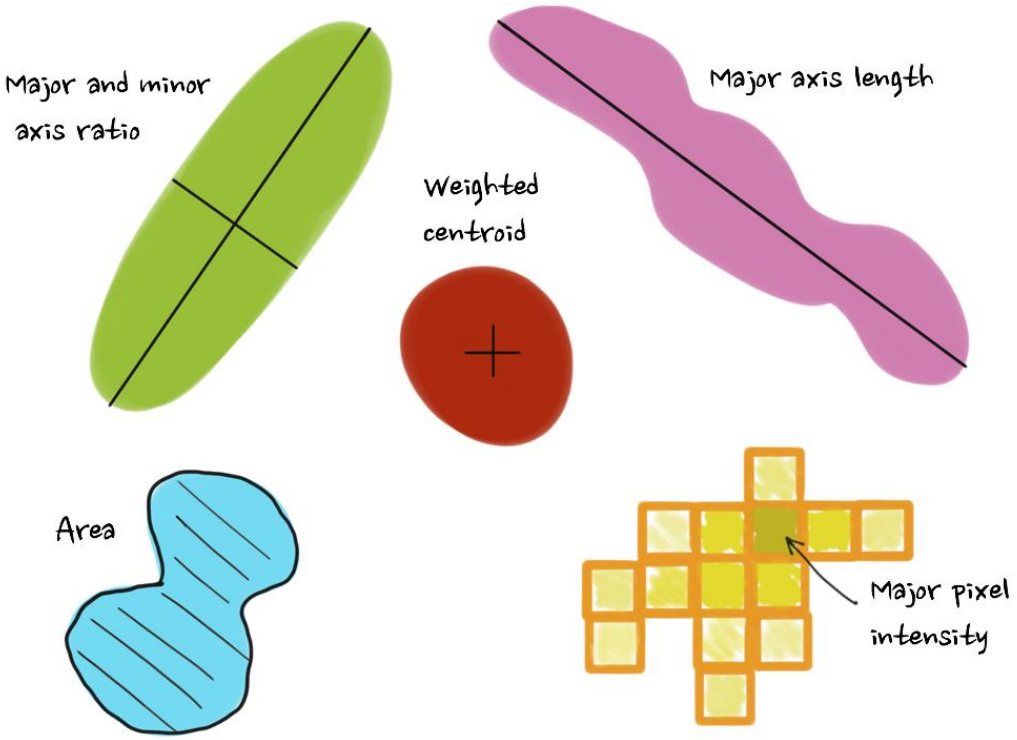


Figure 4.11. Graphic representation of all of the extracted features

Moreover, another modification that was included in this new version is the use of the *regionprops* command to compute the area instead of the customized program previously used. The reason behind this decision is that the customized technique was very computationally expensive and therefore the code took longer to compile all experiments. Also, it was considered that the results offered by the customized algorithm offered a very elevated precision that it wasn't in fact needed for our code. The *regionprops* command offered an already good enough area approximation.

In the previous code, the features extracted were only later used for the tracking process. However, the newest version also uses some these characteristics to perform an object filtering stage that has been added in the segmentation phase and for another trajectory filtering that has also been added to the last step of the algorithm (trajectory analysis).

As previously seen, during the preprocessing step it has already been included an intensity thresholding technique to remove and filter noise from the image. However, this technique is not enough to get rid of all the background noise present in the images. Therefore, the watershed method detects some of these noise as mitochondria. This problem contributes to the more general

problem that was originally explained, the overdetection of mitochondria. To solve this, the new algorithm includes a stage of object filtering that aims to remove the already segmented noise.

This filtering step is done using two different thresholds regarding two different measured features:

- Minimum area: the minimum area threshold is set to 8 pixels squared, which in reality represents 0.1 μm^2. It is considered that mitochondria are never smaller than this minimum area.
- Minimum maximum pixel intensity: one of the features extracted was the maximum pixel intensity and a minimum threshold for this value was set to 145 (out of the maximum possible value 255). This means that a mitochondrion has to present at least a maximum pixel intensity of 57% of the maximum possible fluorescence intensity.

All objects are examined one by one and if any of them present a smaller area value or maximum pixel intensity than the established threshold values, the object's tag is removed and therefore considered as background and not a mitochondrion. Examples of the results offered by the filtration process can be seen in Figure 4.12. and Figure 4.13. Both of them offer clear examples of the effects that filtering has on the images. However, the first image includes more cases of filtering by size, since it contains many elements so small that even at first sight it can be seen that they are not mitochondria, while the second example is a more representative case of the filtering by intensity technique. There's background noise present that during the watershed stage is segmented. However, it can quickly be seen that in fact these segmented objects are not organelles but rather residual fluorescence on the neural axon. The newest algorithm is able to see this errors and properly correct them.

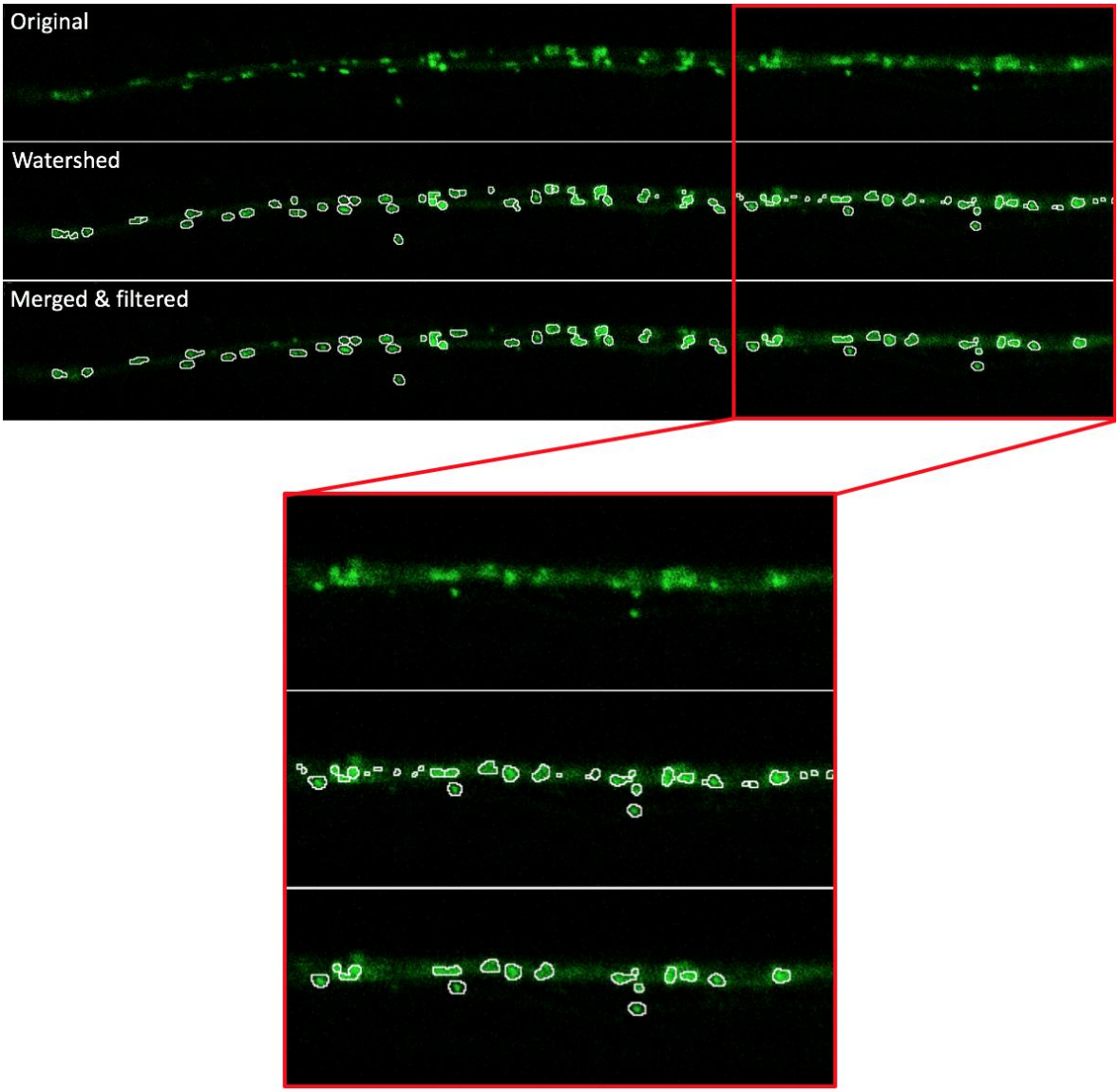


Figure 4.12. Representative example of the results offered by merging and filtering stages. This example includes a good representation of the area filter.

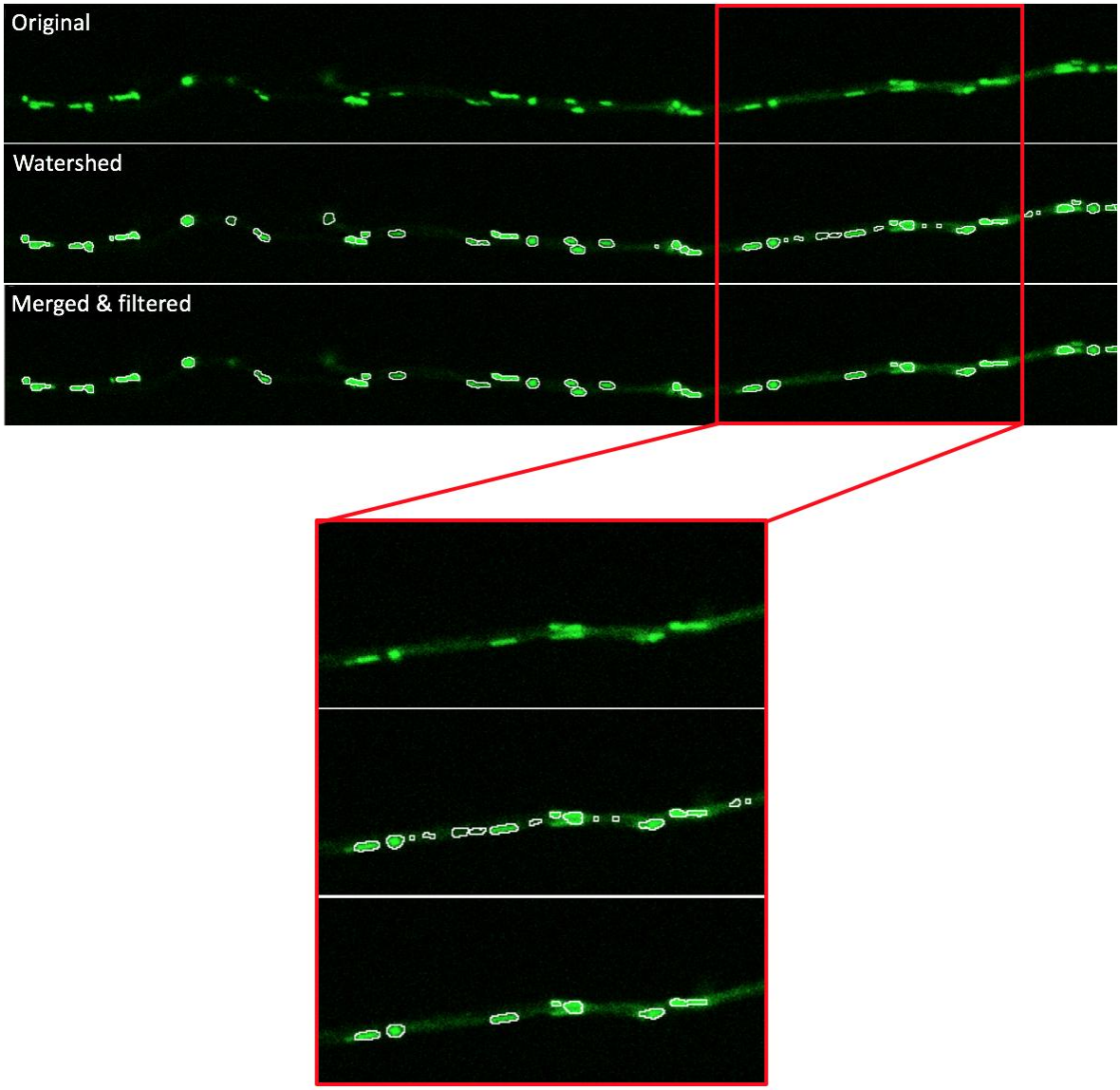


Figure 4.13. Representative example of the results offered by merging and filtering stages. This example includes a good representation of the intensity filter.

After all these filtering techniques applied the presence of segmented noise was significantly decreased. However, the area and the pixel intensity were not the only parameters that were used at first. There was an initial approach that tried to do this filtering technique using parameters regarding the internal structure of the objects.

The original idea was that maybe the remaining noise that had high intensity and area had a different pixel intensity distribution. This analysis was done using the *graycomatrix* command on MATLAB. This command returns four different values [23]:

- Contrast: measures the intensity contrast between a pixel and a neighbor over the whole image.

- Homogeneity: measures the closeness of the distribution of elements in the gray-level co-occurrence matrix (GLCM) to the GLCM diagonal.
- Energy: the sum of the squared elements of the GLCM.
- Correlation: how correlated a pixel is to its neighbor over the whole image.

To do the analysis, a set of 12 different experiments was chosen. Then, from all these experiments, 146 tags were taken and classified as either background noise or real mitochondria. In total, 73 tags were assigned to each group. Afterwards, the four parameters just mentioned were computed for each of the tags from both groups and plots were made to see if these parameters could differentiate noise from real organelles.

These plots can be seen in Figure 4.14 and Figure 4.15. First of all, it can be seen that the parameters of homogeneity and contrast do not vary from the noise tags to the real organelle tags. Therefore, no distinction can be made using this information. Relating to the correlation values, it might seem that there is a significant difference, but the difference is only tangible for a few of the noise particles, therefore is not enough to remove all the noise present. However, it seems that the energy offers a pretty clear difference between the two groups, although there might be some overlapping in the central area. So, a threshold for the energy value was established on the code. However, it was later seen that this threshold removed some real segmented mitochondria in specific cases where the visual distinction wasn't so clear as the distinction seen in the chosen tags for the analysis. Therefore, the energy threshold had to be removed and the only filtering parameters used were the area and maximum pixel intensity.

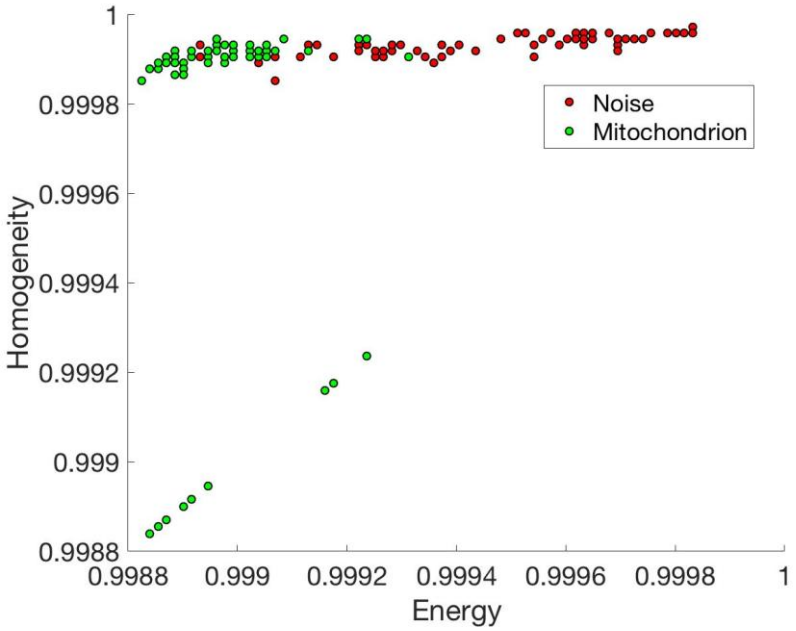


Figure 4.14. 2D plot of the homogeneity and energy values of the tags classified as noise and mitochondria

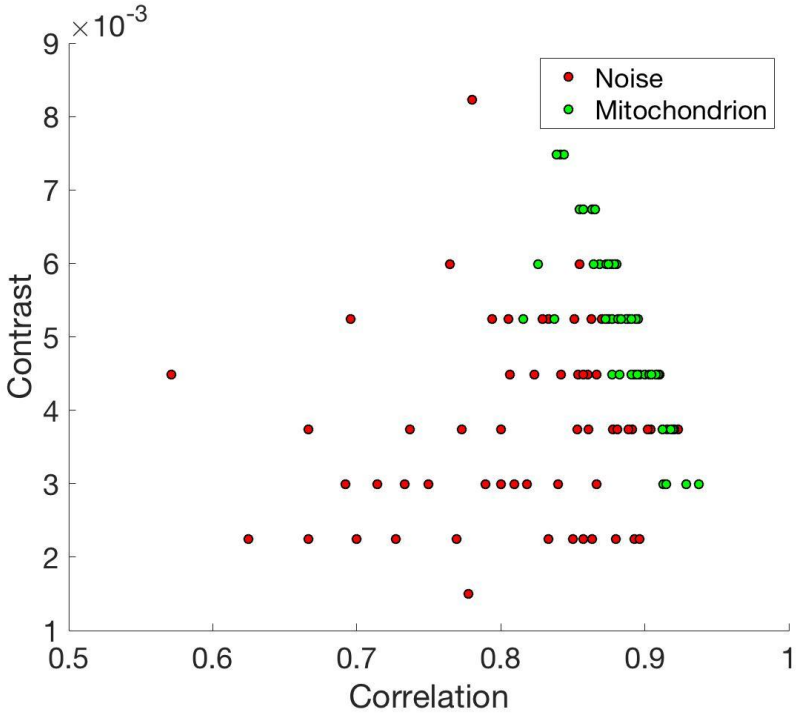


Figure 4.15. 2D plot of the contrast and correlation values of the tags classified as noise and mitochondria

4.2.4. Spatial clustering

4.2.4.1. Original code

In the original code, after the segmentation process, there was a spatial clustering process that was performed in order to find the static mitochondria in the experiments.

To do so, the code performed an X-Y projection of the X-Y coordinates of the centroid of each element in each time step, creating a single image containing all the X-Y coordinates of the detected centroids in all frame in time. In this image, static elements that are not mobile throughout time come out as dense clusters while mobile mitochondria appear as isolated dots.

Afterwards, a hierarchical clustering was performed taking into account the entire set of coordinates in order to differentiate the static elements. It was done by using an Euclidean distance cutoff of $1.2\ m$ (10 pixels) in the y direction (vertical) and $2.4\ m$ (20 pixels) in the x direction (horizontal). The reason behind the difference of distance according to the direction is that in this type of experiments, mitochondria tend to have a soft horizontal vibration that could cause an slight higher variation of the x coordinate of the centroid in the frames.

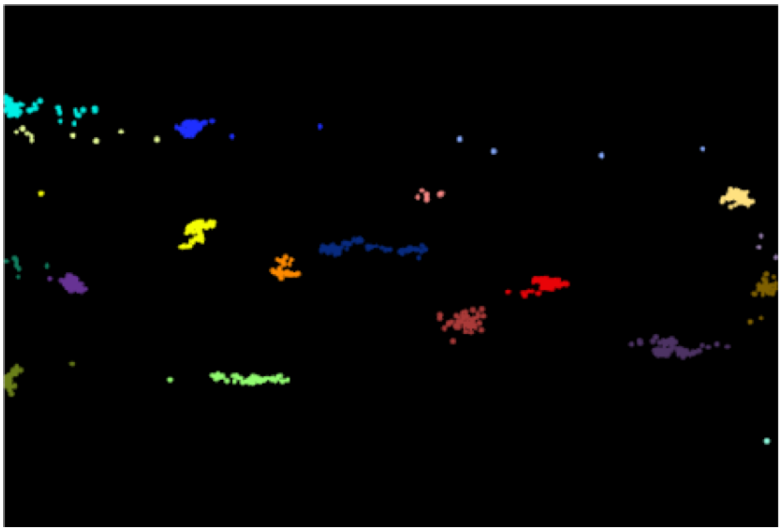


Figure 4.16. Clustering step included in the original version of the algorithm [21].

A representation of the clustering step used can be seen in Figure 4.16. It is a 2D plot representation where all centroids of the detected objects are projected. Moreover, the centroid coordinates have been plotted in different colors for each assigned cluster. As it can be seen, there are clear compact structures that belong to static elements, while there are other isolated points that belong to motile mitochondria.

Finally, after the spatial clustering, the algorithm was able to classify the static objects from the motile ones. Therefore, it was only necessary to apply the tracking measure on the motile elements.

4.2.4.2. New code

The spatial clustering step was profoundly examined. It was considered whether the results offered by this part of the system were reliable or instead suppressing relevant information on a very early stage. As it was already revealed at the beginning of this chapter, it was decided that this step should be removed and instead, apply the tracking process for all elements in all frames, static and dynamic. Afterwards, the results offered by the tracking process could be used to determine whether an organelle was motile or not.

By having this part of the algorithm removed, the second problem of those identified at the beginning of the chapter was solved. However, some modifications in the structure of the program were needed in order to be able to now work with all structures in the tracking process instead of just the ones that survived the clustering process.

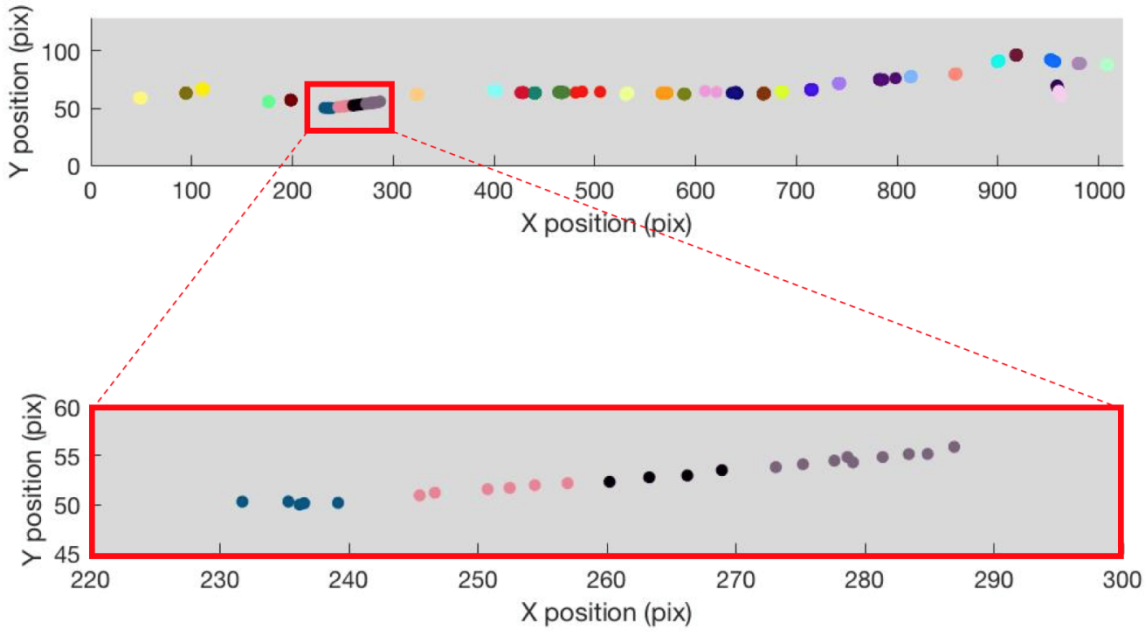


Figure 4.17. Example of a spatial clustering application that offers wrong results

Figure 4.17 shows an example of one of the problems that the clustering stage offered. In the bottom figure it is clearly seen how a trajectory of a motile element is being clustered into four different clusters (color indicates cluster assignment). With the clustering step, this track is considered as four separate static elements, although in reality it's just a single motile mitochondrion. The problem comes from the cutoff distance parameter and the fact that the velocity of this particular mitochondrion is small enough to not create isolated dots but in fact a sequence of dots with small distances between them.

As this example shows, the clustering step was also contributing to the overdetection of mitochondrion, since a real single trajectory was segmented into four different ones.

4.2.5. Tracking

Once all elements have been segmented, it's time to associate the detected elements in each frame with the elements in all the other frames. This way, it is possible to see if a mitochondrion has been travelling over time or not. What this stage wants to accomplish is, if we take a look at Figure 4.18, to be able to see that object A in frame 9 corresponds to object C in frame 10 and same for objects B and D and then be able to see if the centroids of these objects has changed from one frame to the other.

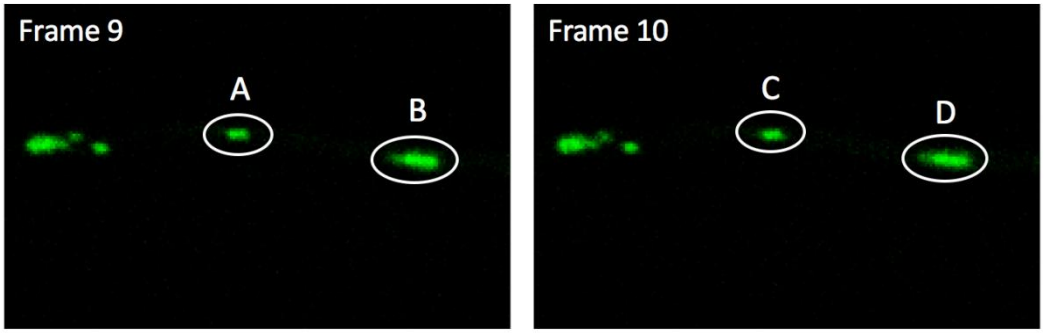


Figure 4.18. Zoom in of two consecutive frames of a specific experiment

4.2.5.1. Original code

In the original code, a customized probabilistic feature-based method was created. This technique uses a Markov model. The area, intensity, position (centroid position) and velocity between consecutive frames of the segmented structures are taken into account.

Specifically, given two objects i and j located in frames k and $k+1$ respectively, the probability that the object i in frame k corresponds to the object labelled j in frame $k+1$ is expressed in terms of the weighted normalized discrepancy of different features of the two objects:

$$p_{ij}^{k} = 1 - \sum_{l=1}^{N_f} \omega_l \, \hat{d}_l \tag{3}$$

In this expression, ω_l is the weight assigned to the normalized discrepancy $\hat{d}_l$ associated to feature f_l. This weight has to be below unity, such that the sum of all weights is exactly unity ($\sum \omega_l = 1$). N_f is simply the total number of features extracted.

The discrepancy d_l is a measure of how much a feature has changed from one frame to the next, and it is defined as $d_l = |f_l(i,k) - f_l(j,k+1)|$, that is the absolute value of the difference of the features between the two frames. The normalization of this discrepancy $\hat{d}_l = d_l / d_l^{max}$ is performed by considering the maximum range a feature can cover from one frame to the next.

In our particular case, we used a total of four features ($N_f = 4$) to characterize each object at each frame: the norm of the 2D position vector of the object within the frame, the norm of the velocity vector measured for the object, the area of the object and the mean intensity of the object. The normalization constants d_l^{max} for each of these features was set to the real extreme values each feature can take. For position and velocity it was set to the distance of the diagonal across the frame. For area it was set to the total area of the frame. For intensity it was set to unity since the feature was already normalized to unity.

Regarding the velocity, it is important to note that this is already by definition a feature that takes into account information in two or more frames. For this reason, this velocity feature is not unique for a particular object in a particular frame. It is a different value depending on against which object it is being compared to. When calculating p_{ij}^{k}, the velocity of object j is $\vec{v}_j = \vec{p}_j - \vec{p}_i$. It is rather the velocity that would be assigned to object j if it corresponded to object i in the previous frame.

Using the model described above, between each consecutive frames all the probabilities are calculated, covering the combinations of all objects in one frame with all objects in the next frame. This collection of probabilities allows to couple an object in one frame to an object in the next frame. The procedure is *top-down*, meaning that we start with the higher probability in the list and couple the two objects involved. Next we continue with the second higher probability value in the list (that does not involve an object that has already been coupled) and so on until we have either no more objects left we reach a minimal probability value.

This minimal probability was a threshold that had to be established in order to allow situations like the entrance and exit of mitochondria from the field of view, the overlapping between organelles and the appearance/disappearance due to movement oblique to the focal plane. Failing to set a minimal probability would lead to for example assigning an object that exits the field of view on one frame to a completely different object that by chance enters in the subsequent frame. The threshold was set at 0.8, but this is a somewhat arbitrary value that highly depends on the normalization used in the tracking method.

The tracking code also included one extra detail that made the technique more robust and reliable than other existing systems. The algorithm has memory, which means that it is capable of saving the information of a given object j in a frame k, up until frame $k + t$. In other words, if an object j appears on frame k and then it overlaps with other elements and therefore is out of sight in the next few frames but reappears afterwards in frame $k + t$, the program is able to link the original object in frame k to the new object on frame $k + t$. This t value is a tunable parameter, but it was originally set to $t = 3$. Therefore, initially the code saved the information of those objects that weren't linked to any objects for the next 3 frames.

It has to be noted that, as mentioned previously, the original version only applied the tracking process to the elements that were not considered static after the clustering process. Therefore, there was only a small portion of mitochondria that would go under this analysis.

On the original version, after doing the tracking of the motile elements, there was nothing left to do. The results had already been created: static objects were found in the clustering step and the trajectories of motile elements was found in this step. Therefore, the code was ended here.

4.2.5.2. New code

After a deep analysis of the results offered by the code, it was seen that the results offered by the automated tracking were really good, and therefore not much could be done to improve this stage. However, one small parameter was tuned. The parameter referred as t in the previous section, which determines for how many frames the information of a non linked object will be saved, was increased to $t = 7$. The reason behind this modification is that it was seen that in some experiments, some organelles would overlap with other elements or go out of plane and lose fluorescence intensity in the images and therefore stay hidden for over 5 or 6 frames. Thus, the new code is able to solve these specific situations.

Finally, as it has already been explained in the previous section, although the tracking technique uses the same lines of code, it has to been noted that the main difference from the original version to the newest one is that before, tracking was only done to those elements that were not classified as static during the spatial clustering and now, because the clustering step has been completely removed, the tracking process is applied to all the elements.

Up until this point, one step was still missing in the new code compared to the results that the original code offered. A distinction between static and moving elements had to be done. To do so, a distance threshold was applied. It was determined that an element that travelled a distance larger than $1.7\ \mu m$ (15 pixels) over all the frames was moving, while smaller travelled distances were considered just as internal vibrations and were classified as static.

4.2.6. Trajectory Analysis

This part of the pipeline is a whole new addition to the new code, the original code did not include this. After taking a look at the final trajectories offered by the system, it was concluded that some modifications could be done in order to improve the results offered by the program. All these modifications have been included in the last stage of the code, the trajectory analysis.

Mainly, the purpose of this part of the code is to solve some confusions and ambiguities that the algorithm might have experienced during the tracking stage and offer reasonable and reliable solutions. In general, two main problems were commonly found in the resulting trajectories:

- Incorrect allocation of points in trajectories
- Fragmentation of a single trajectory into multiple trajectories

Correction of misallocated points

The first issue that needs to be solved, consists on an incorrect assignment of objects in a trajectory that is not its real trajectory. This problem comes from the fact that objects experience high feature variations in time due to the confocal technique used for imaging. The constant movement of the organelles can cause them to move out of plane and therefore the resulting intensity in the images varies. A change in an object's intensity in the image can lead to a change of most of its extracted features: area, centroid, intensity, major axis length and ratio between major and minor axis.

All these fluctuations mislead the probabilistic feature-based tracking algorithm causing a mix-up, because the same object can change completely from frame to frame and the algorithm is not able to match the object properly. This problem can lead to an incorrect allocation of points that in fact belong in a certain trajectory to an another different trajectory. This incorrect distribution of points might create false moving trajectories, when in fact none of the elements involved are motile but rather static.

To fix this ambiguity, it was decided to use the idea of the Mahalanobis distance. This distance is computed between a specific point in relation to a whole group of points. The distance between the point and the mass is computed not only taking into account the center of masses of the group of points, but also the relative distribution of the points inside the cluster.

In numerical terms, the Mahalanobis distance, $D_M(\vec{p})$, between a given point with coordinates $\vec{p} = (p_x, p_y)$ and a set of points with centroid $\vec{\mu}_c = (\mu_x, \mu_y)$ is computed as:

$$D_M(\vec{p}) = \sqrt{(\vec{p} - \vec{\mu})^T S^{-1} (\vec{p} - \vec{\mu})} \quad (4)$$

In this equation, S is the covariance matrix of the set of points.

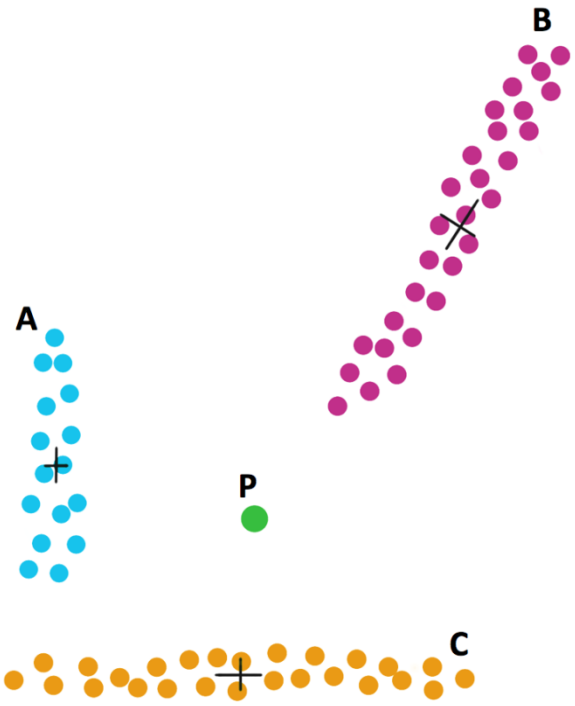


Figure 4.19. Visual aid to explain the Mahalanobis distance.

As an example, let's take a look at Figure 4.19, where three different sets of points (A, B and C) with their respective center of masses and an isolated point P appear. If we were to compute the Mahalanobis between point P and A, B and C, we would see that although the distance between point P and the centroids of A and C is smaller than the distance between P and the centroid of B, the Mahalanobis distance is smaller between P and centroid B. That's because of the distribution of the points inside the sets.

So, in other words, if we were to classify P as part of one of the three sets using the Mahalanobis distance idea, it is more probable that the point will belong to set B than set A or C. We could say that each set presents a certain orientation or distribution. Set A could be considered completely vertical, C horizontal and B diagonal. Because of these orientations, it is difficult to believe that point P would belong to either A or C, but instead falls in a similar orientation than points in B.

Well, the same idea is applied to the resulting trajectories that the program offers. The Mahalanobis distance is computed for each point allocated to a moving trajectory with respect to each static trajectory in the experiment and to the moving trajectory it has been initially assigned to. If any of the distances computed in relation to the static trajectories is smaller than the one in relation to its current trajectory, the point is reallocated to the corresponding stationary track.

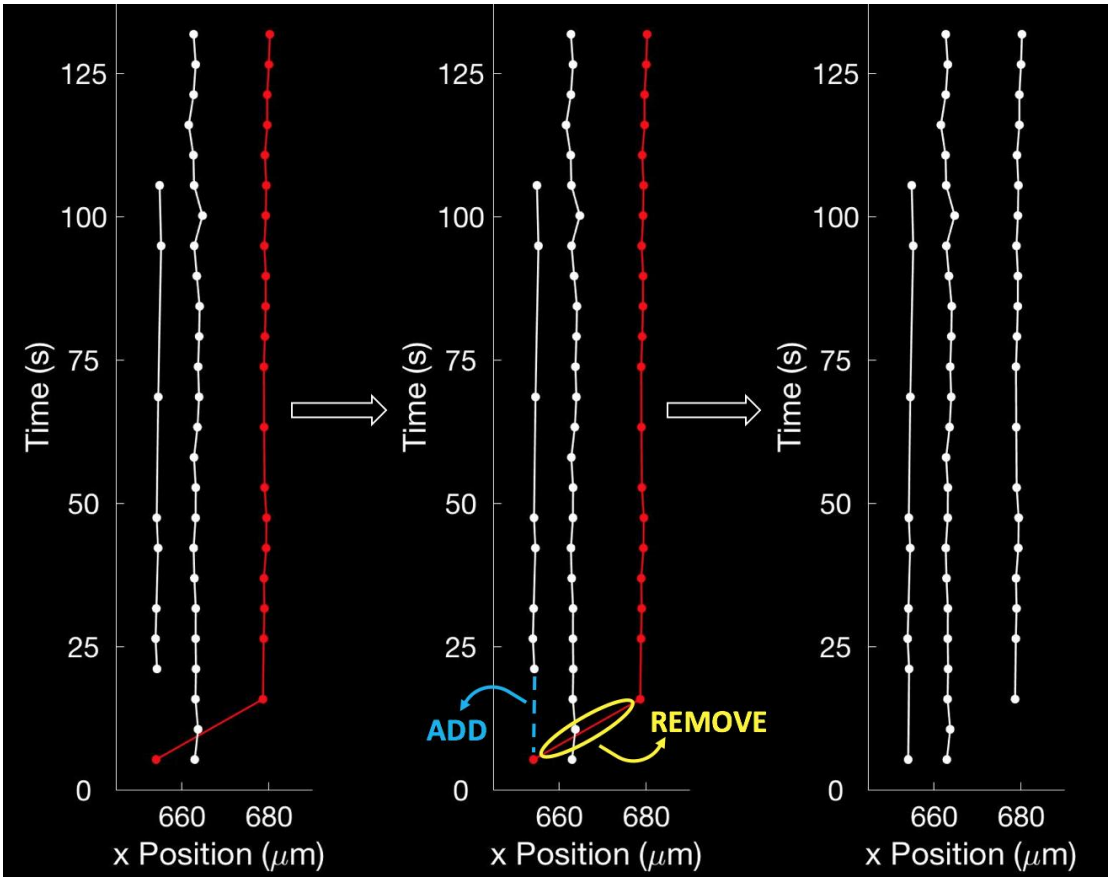


Figure 4.20. Example of the use of the Mahalanobis distance for our trajectories

Figure 4.20 shows an example of the use of the Mahalanobis distance in the algorithm. In this case, in the initial trajectories it can be seen that it is strange that the first point allocated to the moving trajectory (in red on the right) fits perfectly with the centroid of the static trajectory on the left. Moreover, the presumed moving trajectory is static through the rest of time steps. This brings one to think that a mistake was done during the tracking process, and in fact that initial point belongs to the track on the left instead of the right one.

As it can also be seen in Figure 4.20, after this process, a reclassification of the trajectories (as static or motile) is necessary, given that some of the tracks that had been previously considered as motile might have been turned into stationary, as it happens in the example.

So, using the Mahalanobis distance we can solve cases were points have clearly been wrongly allocated. Therefore, there's now only one last problem left to solve: fracture of a single trajectory into multiple ones.

Fix fractured trajectories

The reason behind the fracture of trajectories is due to a multiple segmentation of a single object in a given frame that has not been resolved by the solution previously introduced. An example of what happens to the trajectory is shown in Figure 4.21 (left).

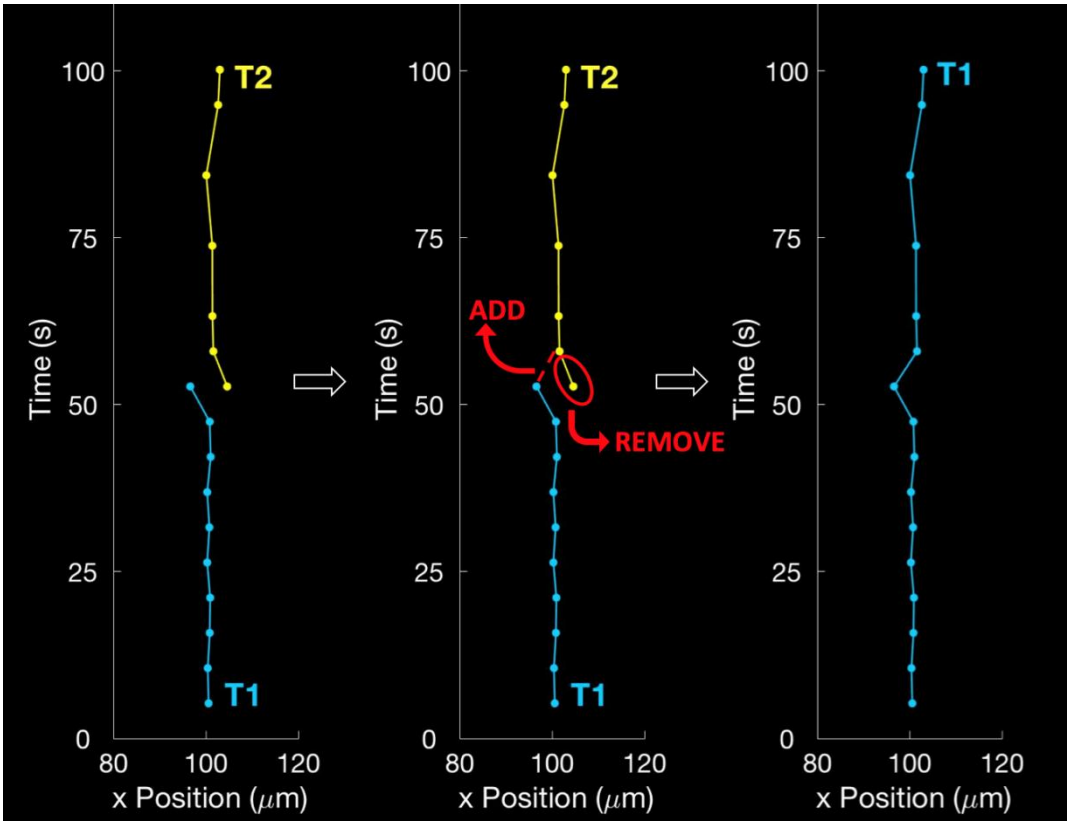


Figure 4.21. Example of a fractured trajectory (left) and the solution offered (middle & right).

In order to solve the ambiguity, the start points and endpoints of all trajectories are considered. If an endpoint and start point occur on the same frame (as it can be seen in Figure 4.21) and they are located within a small Euclidean distance ($\leq 2.2\mu m$), we consider that both points are referring to the same mitochondria that has wrongfully been segmented into two different elements. Then, in order to merge both trajectories, it is necessary to remove one of the segments that the object has been segmented into, meaning the endpoint of T1 or the start point of T2, in the example. It has been established that the one that is removed is the segment that presents less fluorescence pixel intensity. Afterwards, both trajectories are merged into one, offering a final unique trajectory.

One additional step is to perform a reclassification of the trajectories into moving and static again, because this last process might have altered the total travelled distance by the track. Moreover, a new parameter is introduced to do this final classification. The maximum velocity of the mitochondrion along the trajectories is considered. It has been established that any mitochondria

that doesn't have a maximum velocity of at least $0.1\mu m/s$ can't be considered as a moving trajectory because it doesn't have a clear motion pattern.

Extraction of biomarkers

Finally, the last step is to extract information out of the final trajectories, that in fact represent real static and dynamic mitochondria in the experiment. On one side, a feature extraction is performed for each track. These features, often referred as biomarkers, will not be useful for our algorithm, since this is the last step and no further computations will be done, but instead they will be useful for scientists at Sant Joan de Déu, who want to have this information in order to draw conclusions about the mitochondrial behavior. The biomarkers were:

For static and dynamic tracks	Mean area over time Mean major axis length over time Mean ratio between major and minor axis length over time Maximum maximum pixel intensity over time
For dynamic tracks only	Mean velocity over time Total distance travelled over time Direction of movement (anterograde or retrograde)

Table 2. List of the biomarkers extracted for each static and dynamic track

The properties obtained from all tracks were just computed by using the extracted features computed by the *regionprops* command during the segmentation process and averaging that information over time. However, the information collected only for the dynamic tracks had to be computed from scratch. The mean velocity v_m was only computed on the x direction, since it's the one that has an important physical meaning associated as:

$$v_m = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_{i+1}-x_i}{f_{i,i+1}}\cdot\frac{DX}{DT}\right) \tag{5}$$

In this equation, n is the number of time steps where the object has been located, x_i is the x-coordinate position of the object's centroid in time step i, $f_{i,i+1}$ is the number of real frames that exist between time step i and $i+1$ and DT, DX are the temporal and spatial resolution of the images, respectively. The values of $DT$and DX are, as specified in the materials section, 5.27 seconds/frame and 0.12 μm/pixel, respectively.

On the other side, to compute the total distance travelled by each element, two different approaches were done. First, the total range of movement d_r over time was computed as:

$$d_r = x_{max} - x_{min} \quad (6)$$

Where x_{max} and x_{min} refer to the maximum and minimum x-coordinate position of the object's centroid over time.

The other approach used to compute the total distance travelled was the accumulated travelled distance d_a, described as:

$$d_a = \sum_{i=1}^{n} (x_{i+1} - x_i) \cdot DX \quad (7)$$

Finally, the direction of movement was decided taking a look at the mean velocity. If for a given mitochondrion $v_m > 0$, it meant that the organelle travelled mainly from left to right (from the soma to the dendrites), which indicates an anterograde movement. Whereas if $v_m < 0$, it meant the object travelled from the dendrites to the soma and therefore it's a retrograde movement.

One last parameter that was computed was the percentage of movement p_m detected in each experiment, estimated as:

$$p_m = \frac{m}{m+s} \cdot 100 \quad (8)$$

Where m and s refer to the number of dynamic and static mitochondria in the experiment, respectively.

5. Results & Validation

In order to validate our pipeline, we thought it was best not only to carry out a comparison with the original algorithm and the GT data, but also against the results offered by an already published method. This way, we can prove that our program is useful for the set of experiments it was created for but also that it isn't another program similar to the ones that already exist, but it offers improvements that had never been introduced in the scientific community before.

5.1. Evolution of our system

As it was mentioned at the beginning of this report, the main goal of this project was to improve the already existing system that had been previously created and presented several weaknesses. After fixing all the detected flaws, there was a significant change in the results offered by the program.

The following figures show the evolution of the results between the two versions in a kymograph representation. A kymograph is a graph that includes all the detected trajectories in a given experiment. In this case, a 2D version is used, which means that it only plots the variation of the x position (x axis) along time (in frames here, on the y axis). Moreover, in this kymographs trajectories that have been classified as moving have been colored in red, while the static ones are shown in black. Finally, the first frame of the image sequence is shown on the bottom, in order to have a general idea of the experiment that is being graphically represented.

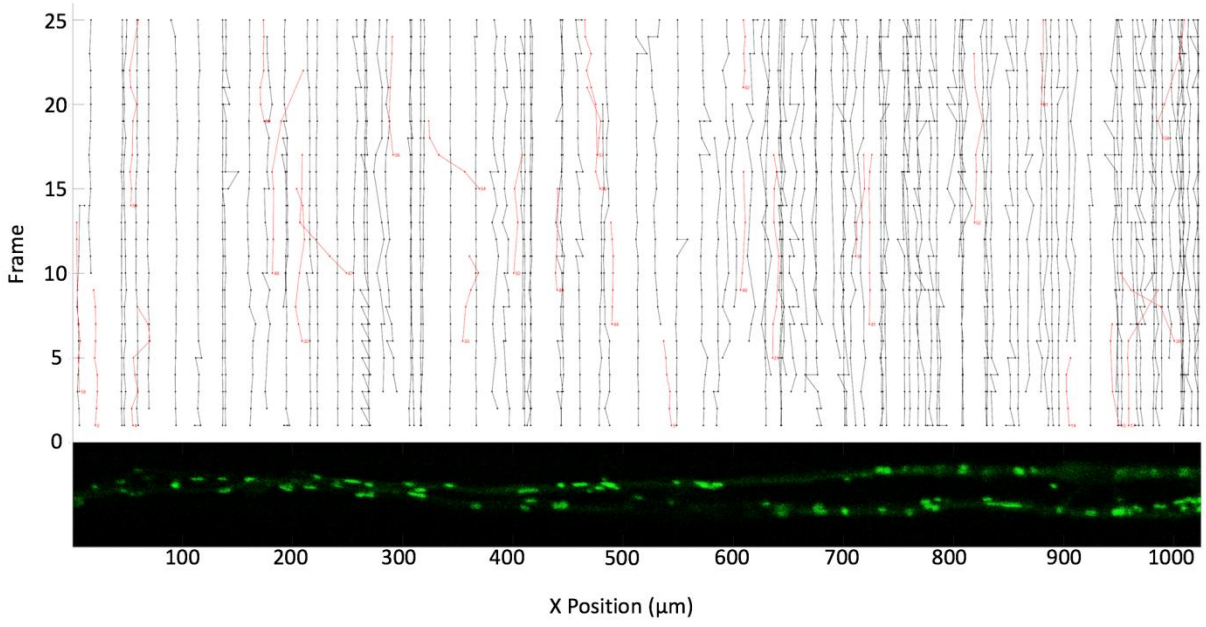


Figure 5.1. 2D Kymograph (only includes x-position over time) of EXP110 resulting from the original program

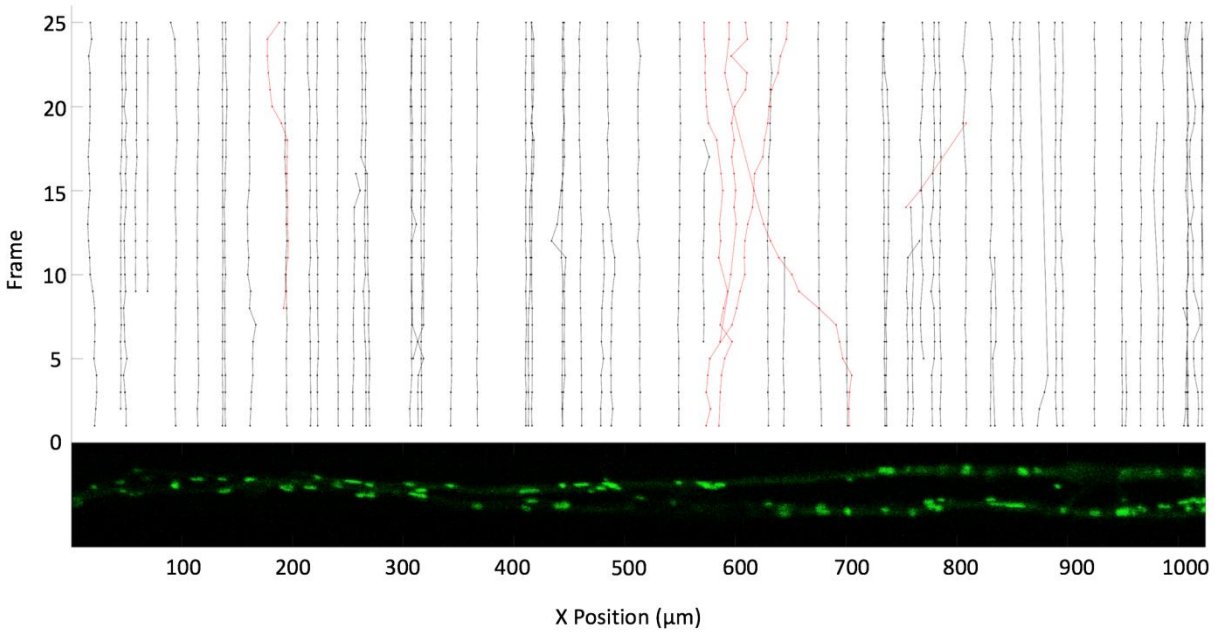


Figure 5.2. 2D Kymograph (only includes x-position over time) of EXP110 resulting from the new program

As we can see on Figure 5.1, the final data provided by the original pipeline is very overcrowded, with broken trajectories, trajectories marked as motile that don't show any type of movement, etc. However, the final data provided by the newest program (note that it's the same experiment as for the original version) shows a much less crowded population, with continuous trajectories that are much more similar to the reality of the experiment. Moreover, all motile trajectories present some type of displacement.

Moreover, another interesting comparison parameter is the total number of detected mitochondria overall the available experiments. If we take a look at it, we can clearly see the significance of the modifications done on the program. Table 3 shows these information, and as it can be seen, there has been a significance reduction in the number of mitochondria detected on the experiments, reducing it almost by half. This is what was expected, since, as previously mentioned, the original code had a problem of overdetection of mitochondria on all experiments.

	Total	Moving	Static
Original Program	5443	903	4540
New Program	3084	520	2564

Table 3. Evolution of the total, moving and static number of mitochondria detected overall the set of experiments available

The following image includes a 3D kymograph version of one of the results offered by the latest algorithm version, in order to properly see the correlation between the trajectories and the initial frame of the experiment. 3D kymographs are available as a result for each experiment examined by

the program, as well as all the other information mentioned: trajectory features included in Table 2 and number and percentage of static and dynamic mitochondria for each experiment.

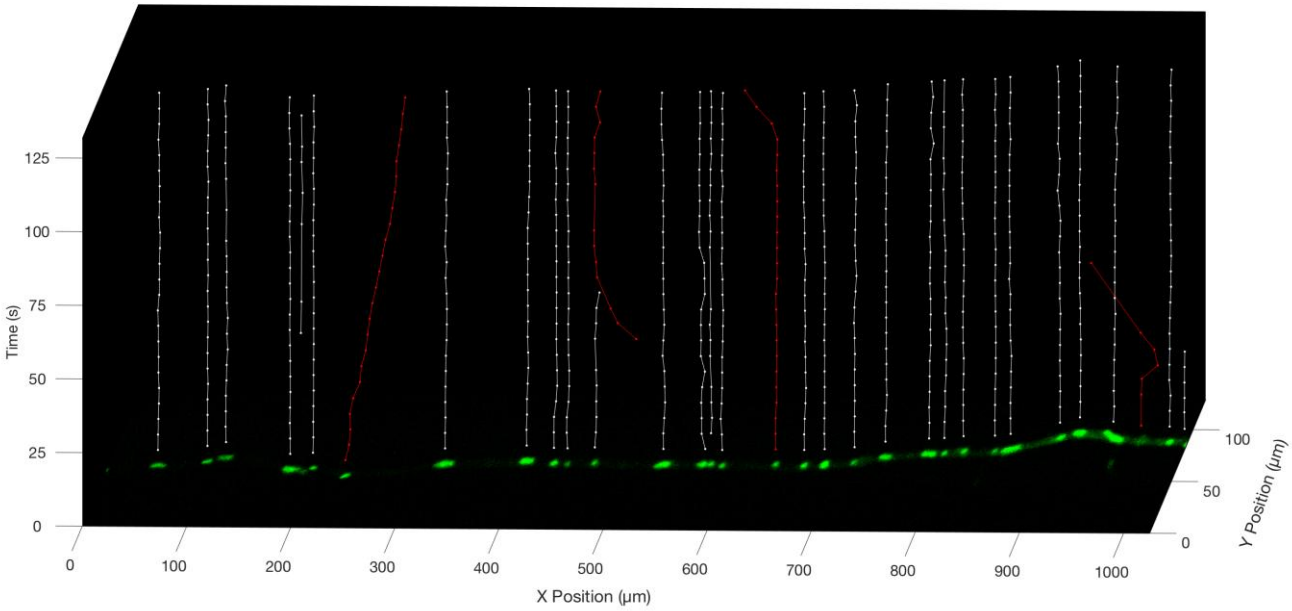


Figure 5.3. 3D kymograph to illustrate the kind of results that the program can offer for each experiment

5.2. Validation against expert's results

Once we have corroborated that the newest version of our algorithm has evolved as expected, it's time to see if it also coincides with the results found by an expert. A few years back a researcher in Sant Joan de Déu was in charge of analyzing all the experiments provided to us. So, we have a GT database that can be used to check if the results offered by the automated system are similar to it or not.

The following table includes the number of detected mitochondria (total, moving and static organelles) by both versions of the code in comparison with the GT values.

	Total	Moving	Static
Ground truth	3026	357	2669
Original Program	5443	903	4540
New Program	3084	520	2564

Table 4. Number of moving, static and total mitochondria found on all experiments by the two versions of the algorithm and GT values.

As it can be seen, the newest results are much closer to the GT values than the original ones. With this information, we can conclude that the final results are a more faithful representation of the reality in the experiments. However, to have a more accurate quantification of the accuracy offered by the oldest and newest version of the code in relation to the GT data, it was decided to compute, for each experiment, the discrepancy between the total, moving and static number of detected mitochondria presented by each method in comparison with the GT. From this, a mean discrepancy and its standard error can be computed as indicated in the following formulas.

$$D_{ij} = N_{ij} - N_{iGT} \tag{9}$$

Here D_{ij} is the discrepancy of the number of detected mitochondria between GT and system j on experiment i. The variables N_{iGT} and N_{ij} refer to the number of detected mitochondria (total, moving or static) on experiment i by the GT and system j, respectively.

$$\overline{D_j} = \frac{1}{k}\sum_{i=1}^{k} D_{ij} \tag{10}$$

In this equation $\overline{D_j}$ is the mean discrepancy resulting between system j and GT. k is the total number of experiments that are being analyzed (in this case 71) and D_{ij} is, as shown above, the discrepancy of detected mitochondria between GT and system j on experiment i.

$$SD_{\overline{D_j}} = \frac{\sum_{i=1}^{k}(D_{ij} - \overline{D_j})^2}{k-1} \tag{11}$$

$SD_{\overline{D_j}}$ is the standard deviation of the discrepancy values over all k experiments. From this, we can compute the standard error associated as:

$$SE_{\overline{D_j}} = \frac{SD_{\overline{D_j}}}{\sqrt{k}} \tag{12}$$

The values of mean discrepancy of both systems and the associated standard error (SE), computed using the formulas just described, are shown in the following table:

	Total		Moving		Static	
	Mean	SE	Mean	SE	Mean	SE
Original Program	34,39	3,15	7,81	0,76	26,73	2,58
New Program	0,83	1,11	2,33	0,88	-1,50	0,90

Table 5. Discrepancy on the detected total, moving and static number of mitochondria between both versions of the system and the GT.

As it can be seen in Table 9, the results are very similar to the ones offered in the previous table. The original version of the code detected much higher number of total, moving and static mitochondria, while the newest version offers results much closer to the GT values. The new version tends to detect more total and moving mitochondria per experiment, while it's a little short on detecting static elements. However, the discrepancies are much more small than in the original algorithm. Therefore, it can be confirmed that the newest version includes an important and noticeable improvement on the results offered.

Finally, one last source of information to validate our final pipeline with the expert's GT is to evaluate the similarity of the percentage of movement detected on the experiments, differentiating between WT and KO specimen.

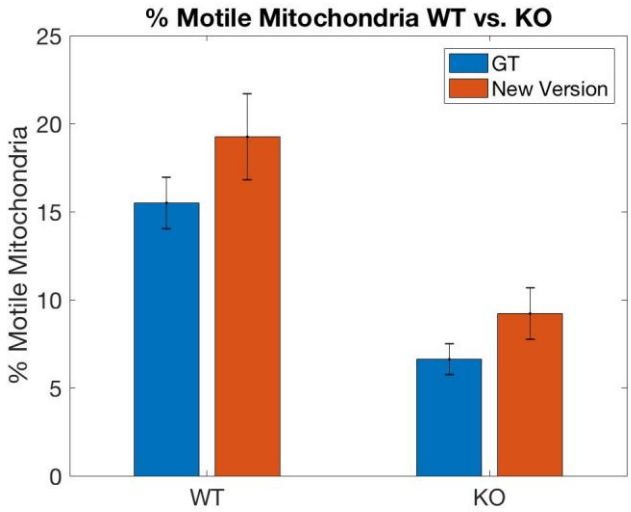


Figure 5.4. Comparison of the results offered by the GT and the newest version of the code with relation to the percentage of movement in each experiment.

As it can be seen in Figure 5.4, although the results offered by the system involve more movement detection, the proportion of movement detection is kept constant between both specimen, which is what is really relevant when extracting significant biological information. Therefore, this shows that the program can be trusted when analyzing differences between both specimen, because although the results are not exactly the same as in reality, they are representative of the difference between WT and KO, which is what is really being studied with this experiments.

5.3. Validation against published software

Although the results of the systems have been validated against a GT provided by an expert, it was decided that it would be useful to compare the code with an already published method. It was chosen to compare it against the software MitoQuant [18]. The software and tutorial for this system

were obtained from http://ese.nju.edu.cn/yogo/mq.zip . The reason for focusing on this specific software to do the comparison is because all the material necessary to execute it was found on the mentioned website and also because it was observed that it worked very similar to our code. By comparing both codes, it will be able to validate our program and have an idea of the performance of the new code in relation to a system that has already been considered good enough by science.

First of all, before being able to do an analysis of the performance of both codes, it was necessary to really understand how the other program worked. The program was available online as a folder of MATLAB scripts and a tutorial that explained the basic steps. It was necessary to see at what the code was really doing in each stage and whether it was different or similar this project's code or not. After taking a deep look into it, a general idea of how the code worked was gained. The code had to be slightly adapted to our images, since it was created to analyze 3D images and ours are in 2D. Also, it was noted that the program included two parameters that could be tuned in order to modify the program's performance:

- Particle enhancement (p)
- Intensity threshold (t)

The particle enhancement parameter was defined in the tutorial as the average length in pixels of the organelles that wanted to be tracked, which in the case of our experiments was 13. However, the intensity threshold parameter was trickier. Therefore, it was decided to create a code that would be able to tell us for which values of particle enhancement and intensity threshold the software MitoQuant performed similarly to our code.

The reason behind this decision is that the only reference that was available, frame per frame, to evaluate the performance of the software were the results offered by our system. That's because the Ground Truth information that we had available, only included the final results of each experiment (number of static and dynamic mitochondria), but not the number of mitochondria found in each frame of each experiment.

To find the ideal parameters for MitoQuant, different values were given to the both adjustable values and it was studied how many elements were detected in each frame for each experiment in comparison to our code. In order to quantify the aptitude of the parameters evaluated, a quadratic error E_{pt}, was computed for each pair of values of particle enhancement, p, and intensity threshold, t, described as follows:

$$E_{pt} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{1}{m_i}\cdot\sum_{j=1}^{m_i}(a_{ij}-b_{ij})^2\right) \tag{13}$$

In this equation, n is the total number of experiments used, m_i is the total number of frames present in experiment i and a_{ij}, b_{ij} are the total number of mitochondria detected by our software and MitoQuant in experiment i in frame j, respectively.

After computing the error for a wide combination of values, it was seen that the minimum error, and therefore, the maximum similarity between both software was found when $p = 1$ and $t = 4$. The representation of the computed error for different values of p and t can be seen in Figure 5.5, which clearly shows that the left bottom corner has the smallest error, which in fact corresponds to $p = 1$ and $t = 4$.

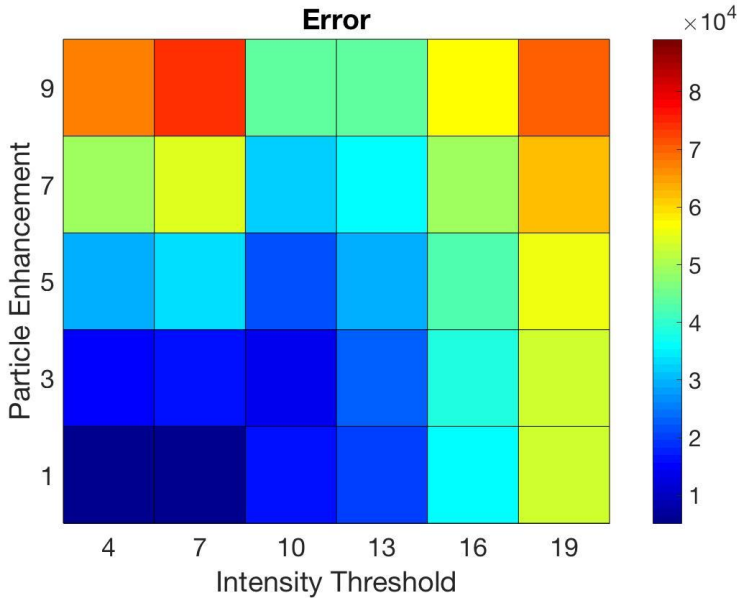


Figure 5.5. 2D plot of the computed error to establish what parameters for particle enhancement and intensity threshold work best for the MitoQuant system

Moreover, to ensure that the selected parameters were in fact the optimum ones to compare both software, a comparison of the detected mitochondria in each experiment was done, to see if they were really taking into account, more or less, the same number of elements in each frame. The results can be seen in Figure 5.6. As it can be observed, the number of objects detected by both systems in each experiment are very similar, which corroborates that the chosen parameters for the software MitoQuant are correct to do the comparison.

In this figure, it can be seen that there are a few experiments that have a total number of detected elements in all frames higher than the rest. This is not because these experiments are more populated than the rest of them, but because they are the 14 experiments mentioned in the materials sections that have been recorded during a longer period of time (5 minutes). Therefore, the total number of frames in the experiment is higher, and because the figure represents the sum of

elements overall all frames, since there's more frames, the resulting value is much higher than the rest.

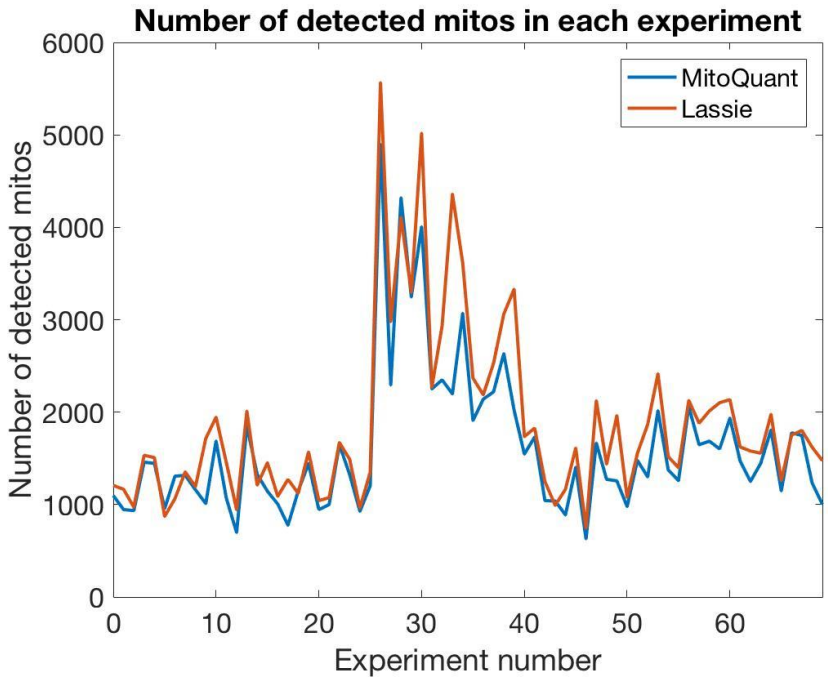


Figure 5.6. Graphical representation of the total number of objects detected on all frames of each experiment (sum of the elements detected in each frame of the experiment) by the two systems.

From this point on, all computations explained and results offered from the software MitoQuant have been used and obtained by taking these specific values of p and t.

So, in order to develop a complete and consistent comparison between both systems, it was decided that the results offered by the algorithms would be tested into two different ways, that would give us different information about the two systems.

5.3.1. Final count comparison

On the one hand, it was decided to compare the total number of trajectories, static and dynamic, that both programs could find when analyzing all the available experiments. That is, running the programs for all the available experiments and compare how many static and dynamic mitochondria each program found. This values, can be compared to the GT information that was provided by researchers at Sant Joan de Déu and therefore, it can objectively be evaluated which system offers a closest approach to reality. Table 6 shows the results of this comparison.

	Total	Moving	Static
Ground truth	2992	351	2641
MitoQuant	3712	172	3540
Our Program	3045	509	3554

Table 6. Number of moving, static and total mitochondria found on all experiments by both systems and GT values.

Note that GT values and the results from the newest version of our code have different numbers in Table 4 than in Table 6 although they seem to refer to the same concept. That is because for unknown reasons MitoQuant was unable to process EXP032 of the list of experiments. Therefore, Table 4 includes the results for a total of 71 experiments, while Table 6 offers the results only for 70 experiments.

As it can be seen on Table 6, for the total number of mitochondria detected by both systems, although both system have a total count higher than the GT data, our program gives a more similar result to the GT values. When taking a look at the moving numbers, both systems have some differences with the GT values. However, the differences are completely different, that's because the software MitoQuant misses a total of 179 mitochondria, while our program detects 158 tracks in excess. So, one is too strict when detecting movement while the other is too lose. However, taking a general look at the information in the table about the number of total, moving and static mitochondria, our program is in general more accurate than MitoQuant.

These type of results are the same as the ones previously seen on Table 4. In that section, further results were included, comparing the discrepancy between the systems and the GT for each experiment. In this section we will follow the same procedure. Equations used here will be the same as the ones explained from equations 9 to 12.

	Total		Moving		Static	
	Mean	SE	Mean	SE	Mean	SE
MitoQuant Program	9,36	1,59	-2,30	0,39	11,66	1,52
Our Program	0,77	1,12	2,29	0,89	-1,52	0,90

Table 7. Discrepancy on the detected total, moving and static number of mitochondria between both programs and the GT.

As expected from the previous information, it is confirmed that in general MitoQuant tends to overdetect the total number of mitochondria per experiment. Regarding the moving organelles, our program tends to detect more movement than the GT information, while MitoQuant is short on it.

Finally, for the static elements, MitoQuant tends to significantly overdetect many, while our program misses a few per experiment.

5.3.2. Trajectory comparison

However, just comparing the total number of detected tracks didn't seem enough. The final number of static and dynamic mitochondria could be mixed up with false noise tracks and therefore, just looking at the final results is not enough to know if the results offered are in fact reliable or not, even if the numbers offered by the system are close to the ones presented by the GT.

So, in order to complement the comparative information, it was decided to do a comparison of the tracking process of both algorithms, to see exactly how well they could trace a real mitochondrial trajectory. However, to do this, first it was necessary to have some real mitochondrial tracks to compare the programs results to, because the GT information available didn't contain this information.

It was decided, that the best way to approach this was to create a program that could allow user intervention and therefore the user could do the tracking of different mitochondria in different experiments. So, initially, there was a first attempt at this program using the MATLAB App Designer. However, after working with it for a while, it was noted that mouse interaction was not supported by the app, and therefore App Designer was not useful at all in order to do a program that would allow a manual tracking of the mitochondria in the experiments.

So, as a second option, the older version of App Designer in MATLAB was considered, which is GUIDE. This tool does in fact allow mouse interactions to be programmed, and so it was useful for our intended purpose. The visual interface of the resulting program created using GUIDE can be seen in Figure 5.7. As it can be seen, it included the following gadgets:

- Browse button: It allows for the user to select the folder where all experiments are contained
- Experiment number: The user types the number of experiments that he/she wants to work with and it the program searches for all the frames contained in that experiment.
- Slider: using the slider, you can move along the different frames of the experiments, that are shown in right above it. Also, the current frame number is shown on the right top corner of the frame image.
- Add / Remove trajectory button: New trajectories can be added or removed.
- Start / Stop / Undo: Once a trajectory has been selected from the track list, these buttons allow to start or stop the tracking of that trajectory and also undo the last step done by the user.

- Save trajectories button: once the tracking of all desired mitochondria have been finished, the collected information can be saved, in order to be able to use it later.

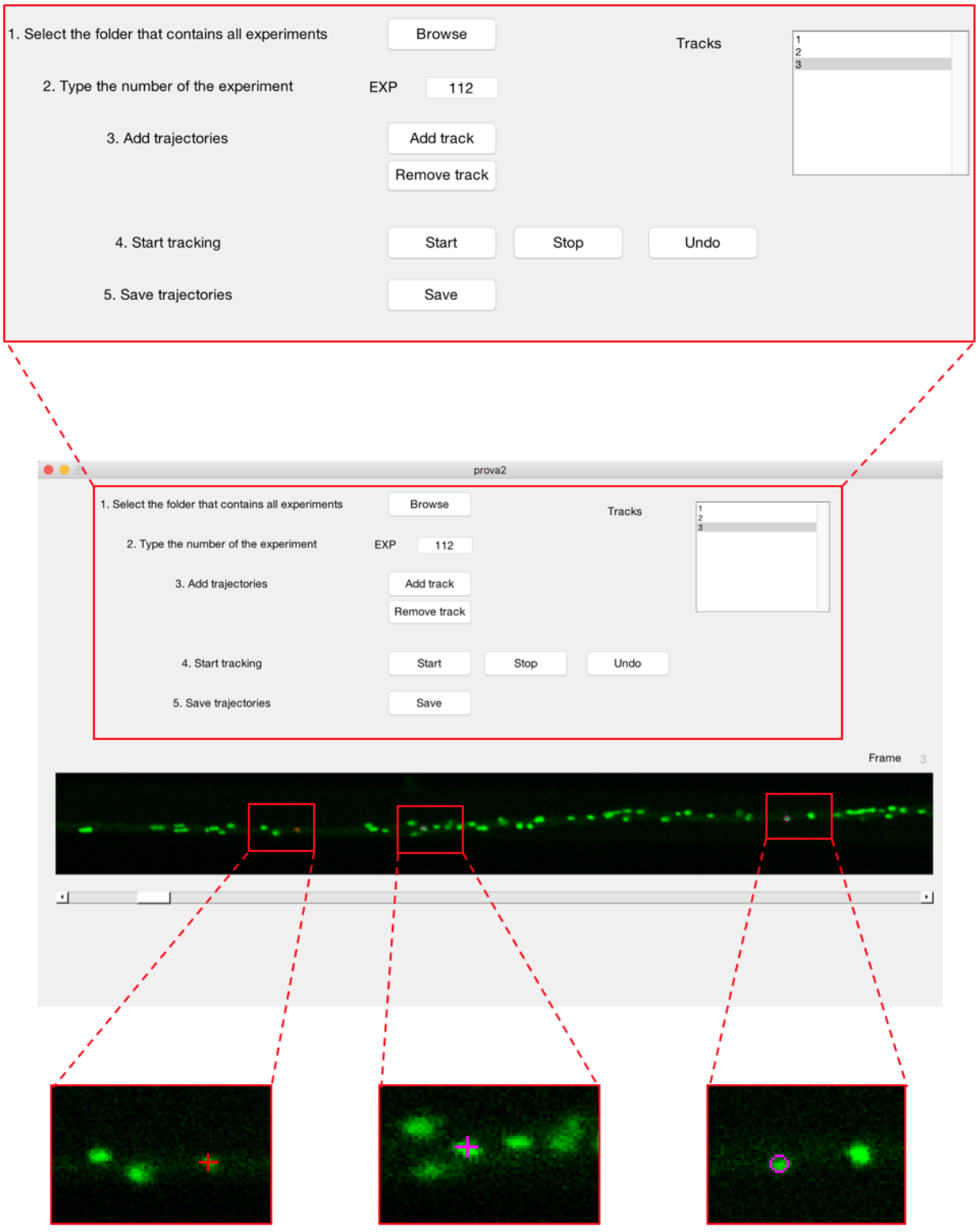


Figure 5.7. Program made with GUIDE used to do the manual tracking of mitochondrial movement

As a result, after doing the tracking for different experiments, the outputs given by this program is a series of *.mat* files, one for each tracked experiment. Each of these files, contains information about one, two, three or whatever number of trajectories were tracked in that specific experiment.

So, after doing an overview of the experiments, 28 experiments were chosen, from each of these experiments between 1 and 4 tracks were selected, which in total gave a final count of 41 real mitochondrial tracks. However, the software MitoQuant isn't able to do the analysis of a specific experiment (EXP032). For this reason, the final number of true trajectories available for comparison was 39. It has to be noted that the criterion used to decide which tracks were the best ones to use was to choose the clearest and easiest to recognize. This means that the chosen tracks, in general, do not present overlapping or lose of intensity due to the mitochondrion going out of plane or focus.

In order to compare the tracking results of both software quantitatively, the track based errors described in [20] was used for each method. The track based error, E^*_{track}, is computed as:

$$E^*_{track} = 1 - \left(\frac{T^*_{correct}}{T_{total}}\right) \tag{14}$$

In this equation, T_{total} is the number of true trajectories (manually tracked) found. In our case, $T_{total} = 39$. Also, $T^*_{correct}$ is defined as:

$$T^*_{correct} = \sum_{i=1}^{T_{total}} \frac{Y_{tracked,i}}{Y_i} \tag{15}$$

where Y_i is the total number of time steps included in the true trajectory i. For instance, maybe a true trajectory has been found only on 10 frames, so the time steps included in this track is 10. This value Y_i is already known thanks to the manual tracking program created. On the other side, $Y_{tracked,i}$ is the number of time steps of trajectory i that have been correctly tracked by the automated system. For example, taking into account the real track that only exists on 10 frames, maybe a program was only able to detect it during 7 frames, out of the total duration of the real track. This value is not known directly and has to be computed.

To compute this value, we created another program with MATLAB. The general idea of this algorithm is, given a true trajectory, find the number of frames where an automated system has detected that same track. The tricky part here is how to determine whether a detected point in a trajectory corresponds to the trajectory that is being studied or not.

Three conditions were established in order to consider that a point found by one of the systems corresponded to a point of the true trajectory that was being studied. First, as obvious as it may seem, the detected point and the real point have to be on the same frame. Second, the point

detected by the system has to be located in a radius of 1.4 μm (12 pixels) in the x-direction and 0.7 μm (6 pixels) in the y-direction from the point of the true trajectory. This condition is set because the manual tracking can include some spatial error due to the fact that the user has to manually select by mouse the centroid of the moving mitochondria. Finally, the last condition is that the point detected by the system has to be part of a trajectory that has at least two more points that can be considered as part of the same true trajectory. Figure 5.8 is a visual aid to help understand which points would be considered correct and which ones wouldn't. In this case, the true trajectory that is being studied is track 1 and it has to be decided whether tracks 2 and 3 include points that correspond to the true trajectory. According to the established conditions, all points from trajectory 2 would be considered as correctly tracked, while none of the points in trajectory 3 would be considered, although there are two points in this trajectory that could seem like they belong to the true trajectory, in fact we can see that they're part of a completely different one and therefore are not considered.

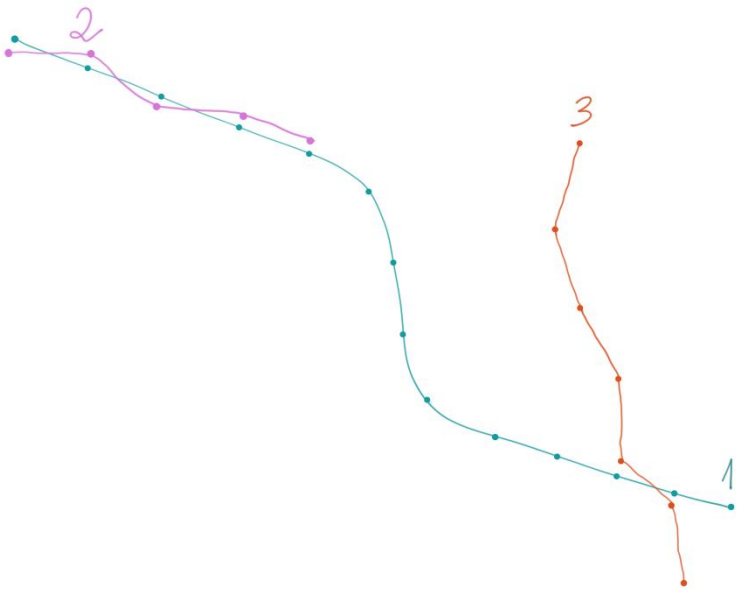


Figure 5.8. Visual aid to understand the functioning of the algorithm that computes

To sum up, the track based error gives an overview of how many points of the real existing tracks the system is able to detect. In other words, it establishes how well the system can track a real mitochondrion along time. The smaller the error, the better the performance of the system. The results for the track based error can be seen on the following table.

	Track based Error
MitoQuant	0.6418
Our Program	0.3784

Table 8. Track based errors for each automated system

As it can be seen, the error is much smaller for our program than for MitoQuant. This means our program is able to more or less correctly track 62% of the trajectories, while an already published method is only able to track 36% of the points and misses the rest.

Finally, one last parameter, very closely related to the track based error was added to the comparison, in order to have more information about the performance of both programs. The parameter was the number of tracks that were detected (out of the 39 GT tracks used). It doesn't matter whether the track has been fully detected (all points have been found) or only a few points from it have been captured. A detected track is one that has been either partially or completely detected. This value is useful when used in combination with the track based error. The same tack based error value can be due to a system that detects completely a few trajectories or due to a system that detects every single trajectory partially. This value helps differentiate between these two cases. The results are shown in the following table.

	Number of tracks detected
MitoQuant	28
Our Program	37

Table 9. Number of tracks detected (out of the 39 GT tracks) by each system

Complementing the results observed in Table 8, Table 9 confirms that our program is able to track individual points and general tracks better than the published software. Here, in this table it can be see that, either partially or completely, our system was able to detect 37 of the 39 GT tracks while MitoQuant was only able to detect 28 of them.

By adding the information given by the last two tables, it can be concluded that our program can track trajectories much better than the already published method.

Figure 5.9 includes a representative example of the results found when comparing the ability of both systems to track the GT trajectories. In this image, it can be seen how well the two automated algorithms can track a given GT track (in magenta). The big dots, in green and black, represent the tracked points ($Y_{tracked}$) found by the two systems. This image clearly shows how our algorithm is able to correctly track 18 points of a GT trajectory consisting of 25 real points, while MitoQuant only detects 8 of these points. Similar to the track based error results, here MitoQuant misses 68% of the trajectory's points, while our program only misses 28%.

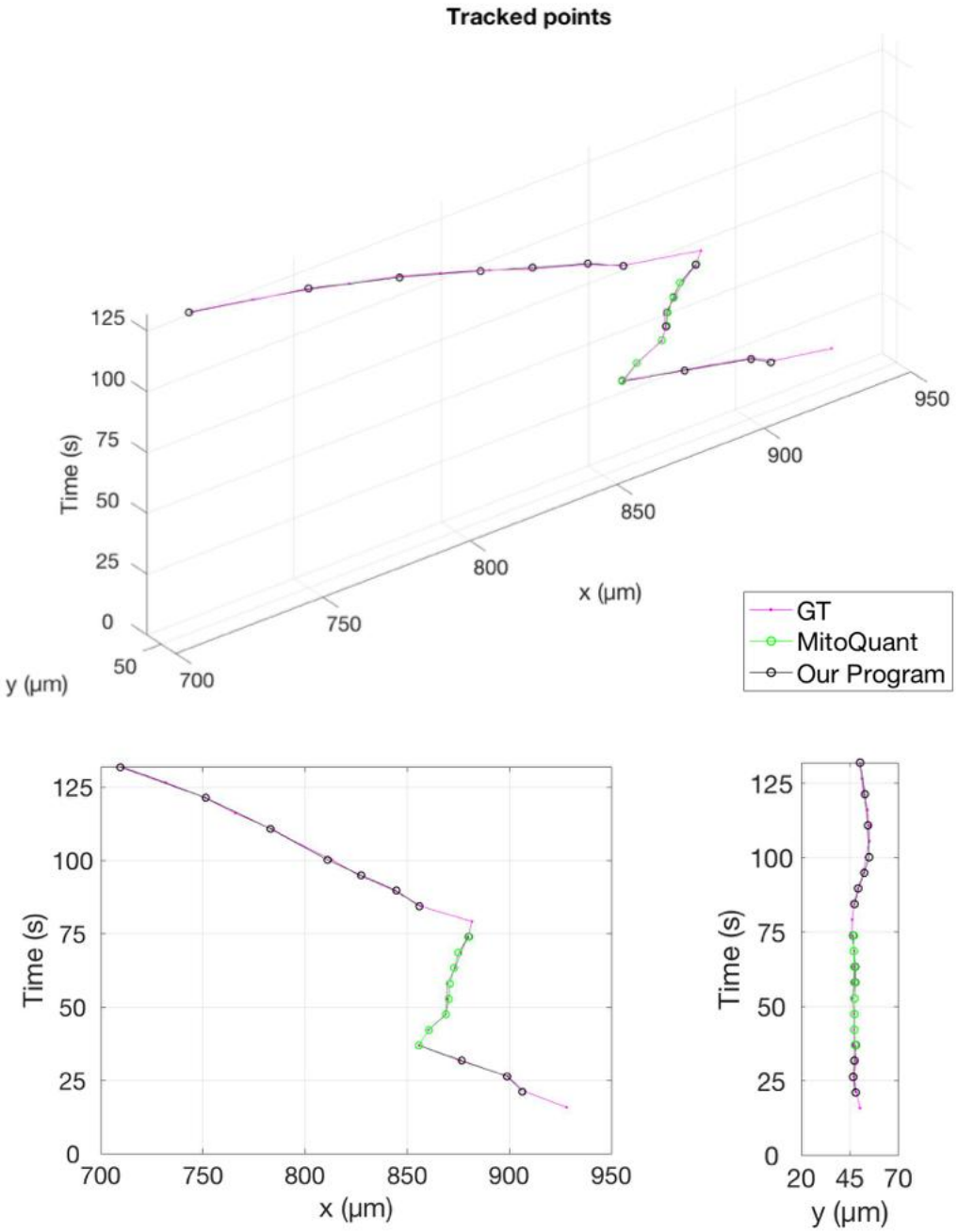


Figure 5.9. Top image is the 3D representation of the tracked points by both systems and GT. Bottom image is the 2D representation on both directions of the same information.

5.4. Extraction of biomarkers

The latest version of the algorithm offered some extra results that were not included in the original version of the code. As already mentioned on previous sections, a features extraction was performed on all the static and dynamic trajectories detected by the algorithm. These features can also be referred as biomarkers, since they are measurable indicators of biological conditions of the organelles.

The biomarkers extraction was included as a request by the scientists in Sant Joan de Déu. For them its crucial to have this information because from it a lot of important conclusions can be drawn. For instance, these results can be used to analyze the relationship between speed and morphological features like size or length, or differentiate between WT and KO mitochondria by these same morphological features, among many other studies.

It has to be noted that this information cannot be extracted manually, and therefore it is a huge progress for the researchers to be able to study these biomarkers, which really allows them to extract important biological information about the organelles that are being studied.

The following figures include a few examples of the features that have been studied. Figure 5.10 shows the differences of movement on mitochondria that are WT and KO. In this case, it can be clearly seen how WT mitochondria present more movement that KO. In Figure 5.11 it can be seen the mitochondria length distribution of WT and KO specimen. As it can be seen, KO mitochondria tends to be longer than WT. Figure 5.12 shows the distance travelled by mitochondria of the WT and KO specimen, differentiating between the anterograde and retrograde directions. As it can be seen, WT travels longer distances than KO for both directions and also the retrograde direction includes longer travelled distances for both specimens. Finally, Figure 5.13 shows the mean velocity presented by mitochondria according to their direction of movement and their specimen. As it can be seen, the only real difference between specimens' velocity exists for the retrograde direction, where WT mitochondria are faster than the KO ones.

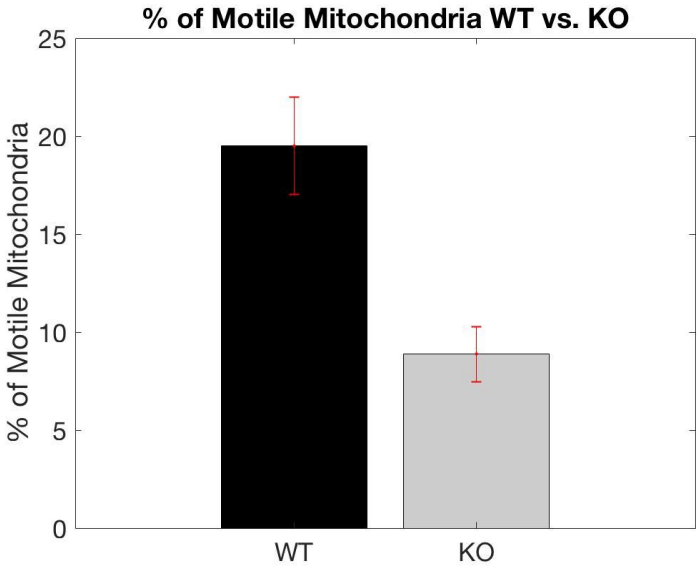


Figure 5.10. Graphic representation of the percentage of motile mitochondria in WT and KO specimen

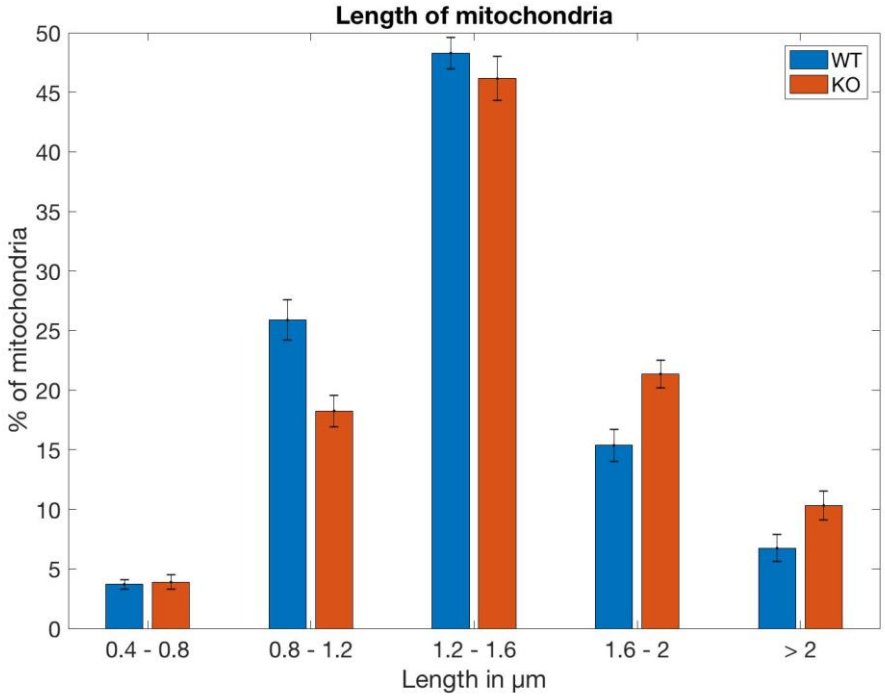


Figure 5.11. Graphic representation of the length of the mitochondria depending on the specimen and classified by length intervals.

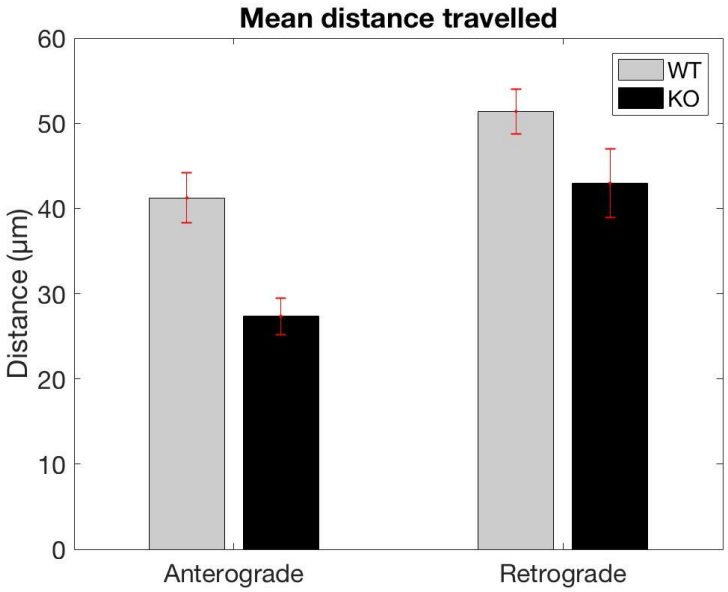


Figure 5.12. Graphic representation of the relation between the distance travelled by mitochondria and the direction of motion, comparing both specimen.

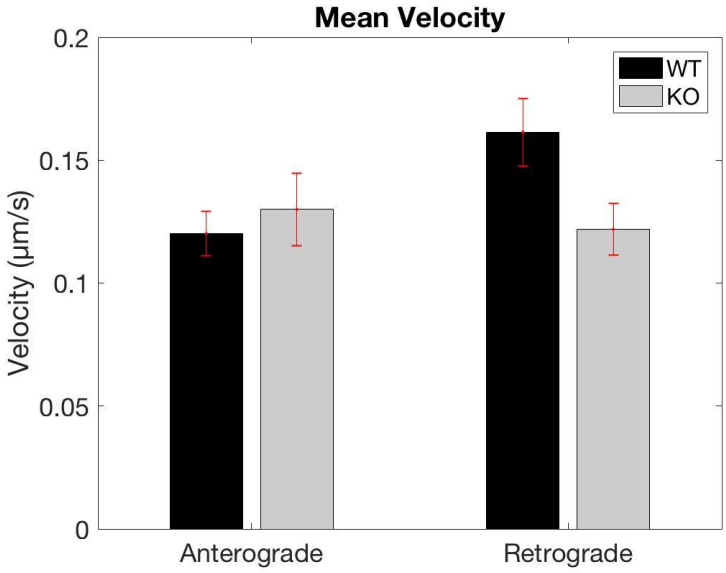


Figure 5.13. Graphic representation of the mean velocity according to the direction of movement and specimen of the mitochondria.

6. Environmental impact analysis

This whole project includes work and tools that have been exclusively developed with the software MATLAB. The only resources needed to carry all this work is electricity and a computer. Therefore, the main environmental impact of this project is related to the use of computers.

Nowadays, modern technology involves many social and environmental problems that are usually unknown or ignored by the consumer. All this problems are presented in the clip created by a group of students from *Ingenieria sin Fronteras*: https://vimeo.com/130808710.

During the manufacturing process of these devices many actions are necessary. On the one hand, the extraction of minerals like coltan in many African countries involves social and environmental problems like slavery, armed conflicts and soil and water contamination. However, the problems don't stop here. Once the device is thrown away, if it's not recycled properly in an official center, it ends up in illegal dumpster in different countries around the world like Ghana, China or India.

The only way to achieve a minimum environmental impact is making us, the users, be more aware of the problems and consequences associated to this type of consumption and try to carry out a more responsible and sustainable usage of technolo

Conclusions

The resulting automated system of this project has accomplished every established goal that was set at the beginning of the project. Flaws that caused unreliable results on the original version have been identified and modified accordingly. These modifications have included the use of different techniques as for example noise filtering, merging of oversegmented objects and tracking analysis. All these changes included in the newest version have significantly improved the results offered when analyzing the available set of experiments coming from Fundació Sant Joan de Déu. The following table includes a summary of all the modifications and additions that have been done throughout the project, divided into subsections.

Preprocessing
Increase Gaussian filter size
Convolution to potentiate horizontal structures
Segmentation
Function to merge oversegmented objects
Filtering of segmented objects
Extraction of new features
Differences between real organelles and noise via pixel distribution
Spatial Clustering
Removal of this step
Tracking
Increase frame memory of the tracking process
Perform the tracking on all elements (static & dynamic)
Trajectory Analysis
Function to detect incorrectly allocated points
Function to fix broken trajectories
Extraction and analysis of biomarkers
Results & Validation
Computation of discrepancies between GT and systems
GUI for manual tracking of mitochondria
Extraction of results from MitoQuant
Program to compute track based error

Table 10. List of modifications and new additions included in each step of the pipeline.

It can be guaranteed that the results offered by the latest system are reliable enough in order to be used in future occasions. This statement is supported by the two different validations that have been performed during the course of the project. Firstly, it has been seen that the program can obtain results very similar to the GT data that was collected by an expert on the field while doing a deep analysis of the experiment images. Moreover, in order to make sure that the program is different to the already existing systems and that it includes new improvements, the program has been compared to an already published system, MitoQuant. The results of this comparison have firmly confirmed that our program can perform, in general, better than the published system. Therefore, our program includes improvements and characteristics that had not been achieved in previous attempts. Taking all of this into account, it can be assured that the resulting automated system is ready to be used whenever a new experiment on mitochondrial trafficking is carried out.

However, as in most cases in life, there's still room for improvements to be done in the future. Further work could focus on decreasing the detection of moving tracks. As it can be seen in the results section, the final algorithm tends to overdetect motile objects. The reason behind it is the infiltration of noise as real moving particles. So, in order to avoid this error, a new technique that was able to differentiate between real and noise trajectories should be developed.

It has to be noted that perfect and completely exact results will probably never be achieved. That is because the image sequences that have been studied in this project present many features that make them particularly complex to analyze, not just automatically by a computer, but even by the eye of an expert. The inconveniences of the images are mainly their low frame rate of 5.27 seconds between frames, the presence of high levels of background noise and overpopulation of mitochondria. It can be seen how, particular experiments that for some reason are less populated and present almost no background noise, can be automatically processed offering very accurate results. On the other hand, experiments with high presence of noise and overpopulation cause a confusion to the program and the performance is not as good.

In general, it can be said that the program is a valid and robust method to quantify mitochondrial trafficking. However, one important remark that is necessary when examining the project, is that the system has been carefully designed to analyze very accurately the images provided by researchers in Fundació Sant Joan de Déu. Therefore, it doesn't guarantee that the system would work as properly as expected for any type of mitochondrial trafficking images, although it includes a series of parameters that can be tuned in order to adapt the code to new images. In fact, the variability between experiments on mitochondrial movement is one of the reasons why so many attempts at these type of automated programs have been performed. So, it is necessary that if at some point in time researchers really want to have a universal algorithm that is able to process all experiments regarding this matter, to create a protocol for the image acquisition techniques. In this way, it would

be guaranteed that all images follow the same pattern and therefore can be examined by the same software. Until this day comes, it is very unlikely that there will even be an algorithm that can be used for all types of experiments of mitochondrial trafficking.

Budget

The budget of this project can be subdivided into two main sources of expense. On one hand, we have the cost of the engineering work, which required several hours for each part of the project. For a junior engineer the cost per hour has been estimated to 35 €/h. All this information has been summarized in Table 11.

Task	**Hours**	**Cost** (35€/h)
Previous study		
Research on imaging techniques	20	700 €
Research on mitochondrial traffic	20	700 €
Research on already existing methods	40	1400 €
MATLAB Code		
Understand existing code	90	3150 €
Solve preprocessing and segmentation problems	150	5250 €
Solve tracking and tracking analysis problems	100	3500 €
Results & Validation		
Extract results from the program	30	1050 €
Check reliability of the results	50	1750 €
Report		
Text development	80	2800 €
Figure editing	50	1750 €
TOTAL		**22050 €**

Table 11. Budget for the engineer work, computed in hours and converted to €.

On the other hand, we have the cost related to the hardware and software licenses used during the project. All details regarding these costs are explained in Table 12.

Item	**Price**
Hardware	
Laptop	1200 €
Office supplies	300 €
Software License	
MATLAB	2000 €
Microsoft Office	149 €
TOTAL	**3649 €**

Table 12. Budget for the hardware and software licenses used in the project.

So, the final cost of the project is 3649 + 22050 = **25699€.**

Publications

As a result of the final outcome of this project, two papers have been written and sent to a couple conferences:

- Cell Symposia Multifaceted Mitochondria, June 4-6, 2018, Paradise Point, San Diego, CA, USA
- IEEE 40th International Engineering in Medicine and Biology Conference, July 17-21, 2018, Honolulu, Hawaii, USA

Both conferences have accepted the papers as Research Posters.

Finally, a manuscript is also being written, containing the whole pipeline and results obtained and is expected to be published in the near future.

Bibliography

[1] L. Wiemerslage and D. Lee, "Quantification of mitochondrial morphology in neurites of dopaminergic neurons using multiple parameters," *J. Neurosci. Methods*, vol. 262, pp. 56–65, Mar. 2016.

[2] "Mitochondrial Function." [Online]. Available: https://biologywise.com/mitochondrial-function. [Accessed: 19-Feb-2018].

[3] T. L. Schwarz, "Mitochondrial Trafficking in Neurons," *Cold Spring Harb. Perspect. Biol.*, vol. 5, no. 6, pp. a011304–a011304, Jun. 2013.

[4] "Questions in Anatomy & Physiology | Socratic." [Online]. Available: https://socratic.org/anatomy-physiology?page=78. [Accessed: 23-Apr-2018].

[5] A. Lajtha and K. Mikoshiba, Eds., *Handbook of Neurochemistry and Molecular Neurobiology*. Boston, MA: Springer US, 2009.

[6] W. M. Saxton and P. J. Hollenbeck, "The axonal transport of mitochondria," *J. Cell Sci.*, vol. 125, no. 9, pp. 2095–2104, May 2012.

[7] K. J. De Vos, A. J. Grierson, S. Ackerley, and C. C. J. Miller, "Role of Axonal Transport in Neurodegenerative Diseases," *Annu. Rev. Neurosci.*, vol. 31, no. 1, pp. 151–173, Jul. 2008.

[8] K. D. Vernon-Parry, "Scanning electron microscopy: an introduction," *III-Vs Rev.*, vol. 13, no. 4, pp. 40–44, Jul. 2000.

[9] K. Stanislav, "Light Microscopy in Biological Research," *Biophys. J.*, vol. 88, no. 6, p. 3741, Jun. 2005.

[10] K. A. Myers and C. Janetopoulos, "Recent advances in imaging subcellular processes," *F1000Research*, vol. 5, p. 1553, Jun. 2016.

[11] Wikipedia contributors, "Fluorescence microscope," *Wikipedia, The Free Encyclopedia.*, 2018. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Fluorescence_microscope&oldid=841196038. [Accessed: 02-May-2018].

[12] "Fluorescence: What, Why, Where and How | Agar Scientific." [Online]. Available: http://www.agarscientific.net/fluorescence-what-why-where-and-how-2/. [Accessed: 01-May-2018].

[13] "Introductory Confocal Concepts | MicroscopyU." [Online]. Available: https://www.microscopyu.com/techniques/confocal/introductory-confocal-concepts. [Accessed: 29-Mar-2018].

[14] "FAQ: What is Deconvolution and Digital Confocal Microscopy." [Online]. Available:

http://www.cyto.purdue.edu/cdroms/micro1/7_spon/vaytek/faq.htm. [Accessed: 01-Mar-2018].

[15] N. H. Kim and Y. Chung, "Automated tracking and analysis of axonal transport using combined filtering methods," *BioChip J.*, vol. 9, no. 3, pp. 194–201, Sep. 2015.

[16] S. Neumann, R. Chassefeyre, G. E. Campbell, and S. E. Encalada, "KymoAnalyzer: a software tool for the quantitative analysis of intracellular transport in neurons," *Traffic*, vol. 18, no. 1, pp. 71–88, Jan. 2017.

[17] O. Welzel, J. Knörr, A. M. Stroebel, J. Kornhuber, and T. W. Groemer, "A fast and robust method for automated analysis of axonal transport," *Eur. Biophys. J.*, vol. 40, no. 9, pp. 1061–1069, Sep. 2011.

[18] M. Chen *et al.*, "A new method for quantifying mitochondrial axonal transport," *Protein Cell*, vol. 7, no. 11, pp. 804–819, Nov. 2016.

[19] J. H. P. Broeke *et al.*, "Automated quantification of cellular traffic in living cells," *J. Neurosci. Methods*, vol. 178, no. 2, pp. 378–384, Apr. 2009.

[20] Lei Yang, Zhen Qiu, A. H. Greenaway, and Weiping Lu, "A New Framework for Particle Detection in Low-SNR Fluorescence Live-Cell Images and Its Application for Improved Particle Tracking," *IEEE Trans. Biomed. Eng.*, vol. 59, no. 7, pp. 2040–2050, Jul. 2012.

[21] A. Vallmitjana, A. Civera-Tregon, J. Hoenicka, F. Palau, and R. Benitez, "Motion estimation of subcellular structures from fluorescence microscopy images," in *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2017, pp. 4419–4422.

[22] "watershed | Scientific Volume Imaging." [Online]. Available: https://svi.nl/watershed. [Accessed: 13-Apr-2018].

[23] "MATLAB Documentation - MathWorks España." [Online]. Available: https://es.mathworks.com/help/matlab/. [Accessed: 10-Feb-2018].

Annex

This section includes some basic parts of the code that has been explained in this report. Not all functions have been included since there are too many and it would be too long. Only the most relevant functions and algorithms have been added in this annex section. However, the entire code is available online at:

https://drive.google.com/drive/folders/1_-zbt8UtnmhTW9BMJ6jdMejhKoIkbmIr?usp=sharing

Code to process all experiments automatically: MainsBatchC4.m

```
%MainsBatchC4
% Perform analysis on all experiments

clear all;close all;clc;
fol='/Users/claudiaserrano/Documents/Claudia/Enginyeria Biomedica/Quart
2017-2018/TFG/Mito3/'; % ruta de les imatges
DX=.1162574;% microns per pixel
DT=5.2739;% seconds per frame
pescar=0;% recicla processat anterior
csvName='EXPS_5.csv';
S=dir(fol);

velth=0.05;
velth=0.1;

control=-ones(length(S),1);
for ii=1:length(S)%
    if(strfind(S(ii).name,'EXP')==1)
        control(ii)=0;
    end
end

tags={'wt','ko'};
% tags={'tif'};
cc=0;
cc=cc+1;parrafada{cc,1}=['ExpName;#Ims;ExpTag;#Stat;#Mov;#Mito;%Mov;#V-
;#V+;'];

for kk=1:length(tags)% recorre tags
    cc=cc+1;parrafada{cc,1}=[upper(tags{kk}) ';'];

AMT=[];AST=[];VMT=[];VST=[];NUM=[];TAT=[];IMT=[];IST=[];MAMT=[];MAST=[];
    NAME=cell(100,1);
    cntE=0;
    disp(' ');
    ensenya([upper(tags{kk}) ' Experiments:']);
    for ii=1:length(S)% recorre experiments
        if(control(ii)==0)
            expid=ii;
            try
```

```
                % mira contingut de la carpeta per veure quin tag es
                R=dir([fol '/' S(expid).name '/']);
                cnt=zeros(1,length(tags));
                for jj=3:length(R)
                    for ll=1:length(tags)
                        if(~isempty(strfind(lower(R(jj).name),tags{ll})))
                            cnt(ll)=cnt(ll)+1;
                        end
                    end
                end

                [num,qui]=max(cnt);
                if(qui==kk)% nomes si hi ha mes arxius del tag que volem
acumular
                    if(num>6)% una mica cutre; comprova que almenys hi ha
6 arxius amb el tag
                        ensenya([ ' Starting ' S(expid).name]) ;
                        control(ii)=1;
                        cntE=cntE+1;
                        if(pescar==1), % pesca dades
                            load([fol '/' S(expid).name
'/Main2_68.mat']);
                        else            % processa exp
                            [AM,AS,VM,VS,TA,IM,IS,MAM,MAS] =
MainC4(S(expid).name,DX,DT);
                        end
                        % acumula dades moving
                        vp=0;vm=0;% counters vplus,vminusp
                        for jj=1:length(AM)
                            %
if(~isempty(find(AM{jj}>=58))),disp(['moving mito ' num2str(jj) '(ind'
num2str(ii) ')']);end
                            AMT=[AMT;mean(AM{jj})];
                            IMT=[IMT;max(IM{jj})];
                            MAMT=[MAMT;mean(MAM{jj})];
                            VMc1=VM{jj}(:,1);
                            VMc2=VM{jj}(:,2);
                            VMc1=VMc1(abs(VM{jj}(:,1))>0.05);
                            VMc2=VMc2(abs(VM{jj}(:,1))>0.05);
                            vx=mean(VMc1);
                            VMT=[VMT;[vx mean(VMc2)]];
                            if(vx<0)
                                vm=vm+1;
                            elseif(vx>0)
                                vp=vp+1;
                            end
                        end
                        % acumula dades static
                        for jj=1:length(AS)
                            %
if(~isempty(find(AS{jj}>=58))),disp(['static mito ' num2str(jj) '(ind'
num2str(ii) ')']);end
                            AST=[AST;mean(AS{jj})];
                            aux=VS{jj};
                            if(isempty(aux))
                                aux=[0 0];
```

```
                            end
                            VST=[VST;[mean(aux(:,1)) mean(aux(:,2))]];
                            IST=[IST;max(IS{jj})];
                            MAST=[MAST;mean(MAS{jj})];
                        end

TATe=[TA,repelem(kk,size(TA,1))',repelem(ii,size(TA,1))'];
                        TAT=[TAT;TATe];
                        NUM=[NUM;numel(AS),vp,vm,numel(AS)+numel(AM)];
                        nomE=S(expid).name;
                        try
                            nom=R(6).name(1:strfind(R(6).name,'_t')-1);
                            catchnom=nomE;
                        end
                        cc=cc+1;parrafada{cc,1}=[nom ';' num2str(num) ';'
nomE ';' num2str(numel(AS)) ';' num2str(numel(AM)) ';'
num2str(numel(AS)+numel(AM)) ';'
num2str(round(100*numel(AM)/(numel(AS)+numel(AM)))) ';' num2str(vm) ';'
num2str(vp) ';'];
                    end
                end
            catch
                control(ii)=0;
            end
        end
    end
    TIT=[upper(tags{kk}) ' experiments (N=' num2str(cntE) ')'];
    save(['mitodataG33' tags{kk}
'.mat'],'AMT','AST','VMT','VST','NUM','TIT','TAT','IMT','IST','MAMT','MAS
T');
    MitoFig4(AMT,AST,VMT,VST,NUM,TIT);
    saveWysiwyg(gcf,['mitodataG3_' tags{kk} '.png']);
    drawnow;

end
if(length(tags)==1),FiguresPaperAdicionals(AMT,AST,VMT,VST,NUM,TIT);end

muntaCSV(csvName,';',parrafada);
csv2html(csvName);

ensenya(['Skipped Experiments: ' num2str(numel(find(control==0)))]);
for ii=1:length(S)%
    if(control(ii)==0)
        disp(['    ' S(ii).name ])
    end
end
close all;
```

Function applied to process each experiment: MainC4.m

```
function [AM,AS,VM,VS,TA,IM,IS,MAM,MAS] = MainC4(fold0,DX,DT)
%FUNCIO

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% DEFINITION OF PARAMETERS %%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

load('mitoparam.mat')

% For Plots & Movie
plt1=0; % Plot after Watershed + rejoin of mitos
plt2=0; % Plot after removal of small areas and noise
plt3=0; % Plot after rawTrack
plt4=0; % Plot for endpoints
plt5=0; % Final kymogram but without image of experiment
plt6=0; % Kymogram
fil=0; % Export movie or not

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% FILES AND FIRST ANALYSIS %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Path where original images are saved
expath=['/Users/claudiaserrano/Documents/Claudia/Enginyeria
Biomedica/Quart 2017-2018/TFG/Mito3/' fold0]; %on estan els experiments

[v0,ims,I,a,b,c,volum]=mitofile(expath,tag);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% PRE - SEGMENTATION TREATMENT %%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[volum,th,templateh]=mitopresegm(volum,gf,ims);

%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% SEGMENTATION %%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%

[volSJ] = mitosegm(v0,volum,th,templateh,intmin,maxsize,bins,kk,plt1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% POST-SEGMENTATION %%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[coords,arees,majorax,intens,volSgn]=mitopostsegm(volum,v0,volSJ,Amin,plt
2,ims,maxintth2,intth);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%% TRACKING %%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[track,IDX,mito]=mitotrack(coords,a,b,plt3,mintracklet,ims);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% POST-TRACKING %%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%

[mov6,sta6,moving6,static6,IDXm6,IDXs6]=mitoposttrack(IDX,mito,track,plt4
,plt5,coords,pixtra,ims,DX,DT,velth,v0);

%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%% RESULTS %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%

[VM,AM,MAM,IM,VS,AS,MAS,IS,alltrack,allpoints,allindex,TA]=mitoresults(mo
v6,sta6,moving6,static6,IDXm6,IDXs6,DX,DT,coords,arees,majorax,intens);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%% MOVIE  %%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if fil==1

mitomovie(a,b,sta6,mov6,static6,moving6,coords,ims,v0,volSgn,IDXs6,IDXm6,
expath,fold);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%% SAVE RESULTS IN FILE %%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

save([expath '/main2_' num2str(maxsize)
'.mat'],'AM','AS','VM','VS','TA');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%% KYMOGRAM %%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if plt6==1

mitokym(mov6,sta6,a,b,ims,v0,expath,maxsize,IDXm6,IDXs6,moving6,static6,c
oords)
end
```

Function used to define all parameters (modifies mitoparam.mat): mitodef.m

```
function mitodef

% For Segmentation
intmin=.15; % Ratio of minimum intensity that a mito has to present
maxsize=68; % Maximum size used in Watershed Segmentation
kk=0.7; % Value used for function joinmito (difference between gradient
intensity and mito intensity)
bins=20; % Used in mitosegm
gf=5; %Used in the definition of the Gaussian filter

% For Noise Removal
Amin=8; % Minimum area of a mitochondria to consider it
intth=65; % Minimum intensity that a mito to consider it
maxintth2=145; % Minimum maxim intensity that a mito to consider it

% For Tacking
mintracklet=3; % Minimum number of frames that a tracklet has to contain
in order to consider it

% For Post-Tracking
pixtra=15; % Minimum number of pixels that a mito has to move in order to
consider it moving
velth=0.1; % Minimum velocity that a mito has to have to consider it a
moving mito

% Files and folders
fold='trackSimple4'; % Name of the folder where files will be saved
tag='ch00'; % Tag that we are looking for in experiments' name

save('mitoparam','intmin','maxsize','kk','bins','gf','Amin','intth','maxi
ntth2','mintracklet','pixtra','velth','fold','tag');
```

Function to load an experiments info: mitofile.m

```
function [v0,ims,I,a,b,c,volum]=mitofile(expath,tag)

% Look for .tif or .jpg images of the given experiment
ims=[dir([expath '/*' tag '*.tif']) dir([expath '/*' tag '*.jpg'])];

% Analyse first frame
[I,cm]=imread([expath '/' ims(1).name]);

% Detect size of the images
[a,b,c]=size(I);

% Detect number of Channels (Color)
ch=1;
if(c>1) % There's more than one Channel (RGB)

[~,ch]=max([sum(sum(I(:,:,1))),sum(sum(I(:,:,2))),sum(sum(I(:,:,3)))]);
end

% Load all frames into one volume (whole experiment contained in one
```

```
matrix)

volum=zeros(a,b,length(ims)); % Volume filled with zeros

for ii=1:length(ims)
    I=double(imread([expath '/' ims(ii).name]));
    volum(:,:,ii)=I(:,:,ch);
end

% Save original volume without modifications
v0=volum;
```

Function for preprocessing: mitopresegm.m

```
function [volum,th,templateh]=mitopresegm(volum,gf,ims)

% Gaussian Filter (2D)

% Define Filter Template
template2=Gaussianeta2d(gf);
template2=template2/sum(sum(template2));

% Filtration - Convolution between volume and filter
volum=convn(volum,template2,'same');

% Definition of intensity threshold to filter each frame in the
Segmentation Section

% Mean intensity of each pixel along t
Vs=sum(volum,3)/length(ims);

% Definition of threshold
th=mean(Vs(:))+1.5*std(Vs(:));

% Definition of a horizontal filter - To boost horizontal structures
after threshold removal
fgh=[3,6,10,6,3];
templateh=fgh/sum(fgh);
```

Function for segmentation: mitosegm.m

```
function [volSJ] =
mitosegm(origvol,filvolum,th,hfilter,minint,maxsize,bins,kk,plt)

%%%%%%%%%%%%%%%%%%%%
%%%%%% INPUTS %%%%%%
%%%%%%%%%%%%%%%%%%%%

%origvol: Original Volum
%th: intensity threshold
%hfilter: horizontal filter
%minint: ratio of minimum intensity
%maxsize: maximum size of a mito
%bins: used in function sizeSegm
%kk: used in function joinmito
```

```
%plt: 0 or 1, either we want to plot all the results or not

numfr=size(origvol,3);

% Definition of volumes that will be filled during Segmentation
volF=zeros(size(origvol)); % Filtred Volume
volS=zeros(size(origvol)); % Segmented Volume
volSJ=zeros(size(origvol)); % Segmented and Rejoined Volume

for ii=1:numfr % For each frame

    % Take each frame
    aux=filvolum(:,:,ii);

    % Filter image using intensity threshold defined previously
    aux(aux<th)=0;

    % Filter image by convoluting with a horizontal 2D filter
    auxf=convn(aux,hfilter,'same');

    % Load each frame of the volume with the filtered frame
    volF(:,:,ii)=auxf;

    % Load each frame of the volume with the segmented frame - Watershed
    volS(:,:,ii)=sizeSegm(auxf,minint,maxsize,bins+8);

    % Load each frame of the volume with the segmented frame - rejoin
    % segmented mitos
    volSJ(:,:,ii)=joinmito2(aux,volS(:,:,ii),kk);

    % Plot
    if plt==1 & ii==1
        figure;clf;set(1,'position',[744.2000  550.6000  979.2000
499.2000]);

cac=uint8(origvol(:,:,ii));cac(:,:,2)=cac(:,:,1);cac(:,:,1)=0;cac(:,:,3)=
cac(:,:,1);

cac1=uint8(auxf);cac1(:,:,2)=cac1(:,:,1);cac1(:,:,1)=0;cac1(:,:,3)=cac1(:
,:,1);
        ax(1)=axes('position',[0 .75 1
.25]);imagesc(enxufatext(cac,'original',[255,255,255]));axis image;
        ax(2)=axes('position',[0 .25 1
.25]);imagesc(enxufatext(cac1,'despres llindar',[255,255,255]));axis
image;
        ax(3)=axes('position',[0 .5 1 .25]);imagesc(volS(:,:,ii));axis
image;
        linkaxes;
    end
        %saveWysiwyg(1,['/Users/claudiaserrano/Desktop/Proves
tfg/Mito3comprovacio/EXP095/trackSimple4/prova1-' num2str(ii) '.jpg']);
end
```

Function for object merging: joinmito2.m

```
function [LJ]=joinmito2(OrigF,SegF,kk)

%FUNCTION DEFINITION
%joinmito is a function that rejoins mitochondrias that have previously
%been segmented into two different regions by the watershed technique,
but
%that are actually the same mitochondria, so they have to be joined

%PARAMETERS
%INPUT
%OrigF: Original Frame (without segmentation)
%SegF: Segmentation Frame
%kk: value to compare mean intensity of gradient and mitos

%OUTPUT
%LJ: Segmentation Frame with joined mitos

frame=OrigF;
L = SegF;
%figure;imagesc(L);axis equal;title('L')

[CC,n]= bwlabel(L>0,4); %Labeling of Original Segmentation all adjacent
tags joined
%figure;imagesc(CC);axis equal;title('CC')

LJ=L; %Creation of the new Segmentation Frame
%TC1=graycomatrix(L,'GrayLimits',[],'NumLevel',length(unique(I)),'Offset'
,[0 1;0 -1;-1 0;1 0;-1 1;1 -1;1 1;-1 -1],'Symmetric',true);
%TC1=graycomatrix(L,'GrayLimits',[],'NumLevel',length(unique(L)),'Offset'
,[0 1;0 -1;-1 0;1 0],'Symmetric',true);
%TC=sum(TC1,3);

for k =1:n
    t=unique(L(CC==k)); %For each tag of CC, to which tag(s) of L
corresponds
    if length(t)>=2 %If a tag of CC corresponds to >1 tag in L
        tg=t; %Tags that are inside a big tag of CC
        allg=0;
        nt=0;
        join=0;
        while allg<length(t)-1 %Number of gradients that have to be found
            if join==0
                nt=nt+1;
            else %join==1
                tg=unique(L(CC==k));
            end
            idxg=0;
            join=0;
            etT=tg(nt); %Tag label that we are analyzing
            [X,Y]=find(LJ==etT); %Image Positon of the tag
            gc=[];
            for j=1:length(X) %For all tag's pixels
                etv=zeros(4,1);
                MP=[0 1;0 -1;1 0;-1 0;1 -1;-1 1;1 1;-1 -1];
```

```
                if 1<X(j) && X(j)<size(frame,1) && 1<Y(j) &&
Y(j)<size(frame,2)
                    for p=1:4
                        idx=[X(j),Y(j)]+MP(p,:); %positions where we have
to look for neigbhours
                        etv(p)=LJ(idx(1),idx(2)); %Tags of these
positions
                    end
                    etv1=etv(etv~=etT); %Delete tag itself
                    etv2=etv1(etv1~=0); %Delet tag=0 (background)
                    for n=1:length(etv2)

gc=[gc;etT,etv2(n),sub2ind(size(frame),X(j),Y(j))];%Index from the tag
etT
                        for m=find(etv==etv2(n))' %position from pix
where neighbour has been found

gc=[gc;etT,etv2(n),sub2ind(size(frame),X(j)+MP(m,1),Y(j)+MP(m,2))];
                        end
                    end
                end
            end
            if numel(gc)==0 %For some reason tags have been previously
joined
                join==1;
                allg=allg+1;
            else
                ng=unique(gc(:,2)); %Number of gradients that have to be
found = Number of tags in contact
                for p=ng'
                    idxg=gc(gc(:,2)==p,3); %Gradients corresponding to
one etT<->one specific neighbour
                    allg=allg+1;
                    idxt1=find(LJ==etT); %Image Position of tag etT
                    idxt2=find(LJ==p);%Image Position of one of the other
tag (neighbour)
                    tint=mean([mean(frame(idxt1)),mean(frame(idxt2))]);
%Mean intensity of both tags
                    gint=mean(frame(idxg)); %Gradient's mean intensity
                    if gint>=kk*tint %Compare gint to tint
                        LJ(LJ==p)=etT;%Join them
                        join=1;
                    end
                end
            end
        end
    end
end

%figure;imagesc(LJ);axis equal;title('LJ')
```

Function for object filtering: mitopostsegm.m

```
function
[coords,arees,majorax,intens,volSgn]=mitopostsegm(volum,v0,volSJ,Amin,plt
2,ims,maxintth2,intth)
```

```
%%%%%CENTROIDE I AREES OBJECTES + ELIMINAR OBJECTES PETITS%%%%%

% Definition of empty arrays of coordinates and areas of each mito
coords=[];
arees=[];
majorax=[];
intens=[];

% Load volumes that will be modified during Post-Segmentation
volSg=volSJ;
volSgn=volSg;

for ii=1:length(ims) % For each frame
    L=volSJ(:,:,ii); % Load each frame

    % Compute area for each segmented object (mito)
    C=regionprops(L,'Area');

    % Remove objects with an area smaller than Amin
    indPetits = find([C.Area] < Amin);
    for etiqueta = indPetits
        [x,y]=find(volSJ(:,:,ii)==etiqueta);
        volSg(x,y,ii)=0; % Objects have been removed from volSg
    end

    % Copy each frame of volSg for volSgn
    volSgn(:,:,ii)=volSg(:,:,ii);

    % Look for all labels in each frame of volSg (remove 0: background)
    labels=unique(volSg(:,:,ii))';
    labels=labels(labels~=0);

    % Compute mean and max intensity of each mito and remove those with
low
    % mean and max intensity (considered as noise)
    for t=labels
        mask = volSg(:,:,ii)==t;
        vt = v0(:,:,ii).*mask;
        mt = mean2(vt(vt>0));
        maxt = max(max(vt(vt>0)));
        stdt = std2(vt(vt>0));
        if (mt<intth && maxt<maxintth2) || maxt<maxintth2
            [x,y]=find(volSg(:,:,ii)==t);
            volSgn(x,y,ii)=0; % Objects have been removed from volSgn
        end
    end

    % Compute Area and Weighted Centroid of all the remaining mitos in
    % volSgn
    L2=volSgn(:,:,ii);

C2=regionprops(L2,volum(:,:,ii),'WeightedCentroid','Area','MajorAxisLengt
h','MeanIntensity');

    % Fill coords and area arrays with Centroid coords and frame and Area
of each mito
```

```
    for jj=1:length(C2)
        if C2(jj).Area>0

            % Centroid Coordenates

fila=[C2(jj).WeightedCentroid(1),C2(jj).WeightedCentroid(2),ii];

            % Areas
            arees=[arees;C2(jj).Area];

            % Major Axis Length
            majorax=[majorax;C2(jj).MajorAxisLength];

            % Intensity
            intens=[intens;C2(jj).MeanIntensity];

            % Centroid coords
            coords=[coords;fila];
        end
    end

    % Plot
    if plt2==1 && ii==1
        figure;clf;set(1,'position',[744.2000  550.6000  979.2000
499.2000]);

cac=uint8(v0(:,:,ii));cac(:,:,2)=cac(:,:,1);cac(:,:,1)=0;cac(:,:,3)=cac(:
,:,1);
        ax(1)=axes('position',[0 .75 1
.25]);imagesc(enxufatext(cac,'original',[255,255,255]));axis image;
        ax(2)=axes('position',[0 .25 1 .25]);imagesc(volSgn(:,:,ii));axis
image;
        ax(3)=axes('position',[0 .5 1 .25]);imagesc(volS(:,:,ii));axis
image;
        linkaxes;
        %saveWysiwyg(1,['/Users/claudiaserrano/Desktop/Proves
tfg/Mito3comprovacio/EXP028/trackSimple4/prova4-' num2str(ii) '.jpg']);
    end
end
```

Function for tracking: mitotrack.m

```
function [track,IDX,mito]=mitotrack(coords,a,b,plt3,mintracklet,ims)

% Index of all mitos
mito=[1:length(coords)]';

% Tracking of all mitos with rawTrack
IDX=rawTrack(coords(:,1:3),b);

% Plot
if(plt3==1)
    for ii=1:max(max(IDX))
        thisc=find(IDX==ii);
        [~,ord]=sort(coords(mito(thisc),3));
```

```
h=plot3(coords(mito(thisc(ord)),1),coords(mito(thisc(ord)),2),coords(mito
(thisc(ord)),3)-1,'.-r');
        set(h,'linewidth',1.5,'markersize',10);
    end
    xlim([1 b]);ylim([1 a]);
    zlim([0 length(ims)]);
    view(-5,50);
end
```

Function for Trajectory Analysis: mitoposttrack.m

```
function
[mov6,sta6,moving6,static6,IDXm6,IDXs6]=mitoposttrack(IDX,mito,track,plt4
,plt5,coords,pixtra,ims,DX,DT,velth,v0)

% Create vector that will contain index of all moving or static mitos in
% array coords
moving2=[];
static2=[];

% Create vectora that will contain index of tracklet to which the mito in
% the same position in vector moving2 or static2 corresponds
IDXs2=[];
IDXm2=[];

% Distinguish moving from static tracklets
for ii=track' % For each tracklet

    % Minmum x coordinate of the centroid in time
    minposx=min(coords(IDX==ii,1));

    % Maximum x coordinate of the centroid in time
    maxposx=max(coords(IDX==ii,1));

    % Absolute distance travelled
    dist=maxposx-minposx;

    % Filter moving from static according to #†pixels travelled
    if dist<=pixtra

        % Save static mito index
        static2=[static2;mito(IDX==ii)];

        % Save static mito tracklets
        for jj=1:length(mito(IDX==ii))
            IDXs2=[IDXs2;ii];
        end
    else

        % Save moving mito index
        moving2=[moving2;mito(IDX==ii)];

        % Save moving mito tracklets
        for jj=1:length(mito(IDX==ii))
```

```
            IDXm2=[IDXm2;ii];
        end
    end

end

% Array with all the tracklet indentificators of moving and static
% separately
mov2=unique(IDXm2);
sta2=unique(IDXs2);

% Compare each point of a moving trajectory with static and moving
centroids
moving3=moving2;
static3=static2;
IDXm3=IDXm2;
IDXs3=IDXs2;

for i=mov2' %For each moving trajectory
    for j=moving2(IDXm2==i)' %For each point in the moving trajectory

        % Centroid of the point
        cmp=coords(j,1:2);

        % Frame of the point
        frp=coords(j,3);

        % Mahalanobis distance from the point to the moving tracklet it
has
        % been associated to
        dm=mahal(cmp,coords(moving2(IDXm2==i),1:2));

        % Give a very high value, so it is higher than the first dist
        % computed
        ds=10e100;

        for k=sta2' % For each static trajectory

            % Mahalanobis distance from the point to the static tracklet
            dist=mahal(cmp,coords(static2(IDXs2==k),1:2));

            % Find if this new distance computed is smaller than the
            % smalles previous distance found
            [ds,ind]=min([ds,dist]);

            % If the new distance is the smallest, save the index of the
            % static tracklet it corresponds to
            if ind==2
                staclu=k;
            end
        end

        % Find the frames already filled with points in the closests
        % static tracklet to the point
        frs=coords(static3(IDXs3==staclu),3);
```

```
        % Find if the point frame is the same as an already filled frame
in
        % the static tracket (Then the point cannot belong to that
        % tracklet)
        comfr=intersect(frp,frs);

        % If the Mahalanobis distance is smaller for a static tracklet
        % than the moving tracklet it had been associated with, and this
        % static tracklet has an empty spot in that same frame, move the
        % point from the moving tracklet to the static one
        if ds<dm && numel(comfr)==0

            % Add index of static tracklet at the end of IDXs3
            IDXs3=[IDXs3;staclu];

            % Add index of point at the end of static3
            static3=[static3;j];

            % Remove index of moving tracklet from IDXm3
            IDXm3(moving3==j)=0;

            % Remove index of point from moving 3
            moving3(moving3==j)=0;
        end

        % Remove elements from arrays that have been put to 0 due to
        % element removal
        moving3=moving3(moving3~=0);
        IDXm3=IDXm3(IDXm3~=0);
    end
end

% Re-compute the new unique moving and static indexes
mov3=unique(IDXm3);
sta3=unique(IDXs3);

% Resort moving trajectories, maybe after removing points now they are
% static

moving4=moving3;
static4=static3;
IDXs4=IDXs3;
IDXm4=IDXm3;

for ii=mov3'
    minposx=min(coords(moving3(IDXm3==ii),1));
    maxposx=max(coords(moving3(IDXm3==ii),1));
    dist=maxposx-minposx;
    if dist<=15
        static4=[static4;moving3(IDXm3==ii)];
        IDXs4=[IDXs4;IDXm3(IDXm3==ii)];
        moving4(IDXm4==ii)=0;
        IDXm4(IDXm4==ii)=0;
    end
```

```
    moving4=moving4(moving4~=0);
    IDXm4=IDXm4(IDXm4~=0);
end

mov4=unique(IDXm4);
sta4=unique(IDXs4);

% Sort static vector after adding new elements to some tracklets
static5=[];
IDXs5=[];

for i=sta3' %For each static tracklet

    %Sort elements by index inside same statick tracklet
    static5=[static5;sort(static4(IDXs4==i))];
    IDXs5=[IDXs5;sort(IDXs4(IDXs4==i))];

end

sta5=unique(IDXs5);

% Join endpoints with initial points

% Consider all tracklets, moving and static
allp=[moving4;static5];
IDX4=[IDXm4;IDXs5];
all=unique(IDX4);

% Create empty arrays
ip=[];
ep=[];
cip=[];
cep =[];

for i=all' %For each tracklet

    % Initial point of the tracklet
    ip1=min(allp(IDX4==i));

    % Plot
    if plt4==1

plot3(coords(allp(IDX4==i),1),coords(allp(IDX4==i),2),coords(allp(IDX4==i
),3),'ko-');
        hold on
    end

    % If initial point of the frame is not 1, save point index and coords
    % (x,y,t)
    if coords(ip1,3)>1
        ip=[ip;ip1];
        cip=[cip;coords(ip1,1:3)];

        % Plot
        if plt4==1
```

```
plot3(coords(ip1,1),coords(ip1,2),coords(ip1,3),'go','MarkerSize',4);
        end

    end

    % End point of the tracklet
    ep1=max(allp(IDX4==i));

    % If it is not in the last frame possible, save point index and
coords
    % (x,y,t)
    if coords(ep1,3)<length(ims)
        ep=[ep;ep1];
        cep=[cep;coords(ep1,1:3)];

        % Plot
        if plt4==1

plot3(coords(ep1,1),coords(ep1,2),coords(ep1,3),'ro','MarkerSize',4);

        end

    end

end

% Plot
if plt4==1
    view(0,0)
end

% Create new arrays with all point indexes and all tracklet
identificators
allpj=allp;
IDX4j=IDX4;
allj=all;

for i=1:length(ip) % For each initial point

    % It hasn't been joined at the beginning to any end point
    joined=0;

    % Remove endpoints that have been removed
    cep=cep(ep~=0,:);
    ep=ep(ep~=0);

    for j=1:length(ep) % For each end point

        % If ip and ep are in the same frame, are close by and ip hasn't
        % been previously joined to any other ep
        if cip(i,3)==cep(j,3) && pdist([cip(i,1:2);cep(j,1:2)])<20 &&
~joined

            % Compute round coordinates of the initial point
```

```
            pip=round(cip(i,1:3));

            % Compute round coordinates of the end point
            pep=round(cep(j,1:3));

            % Tracklet identificator of the endpoint
            jidx=IDX4j(allpj==ep(j));

            % Tracklet identificator of the initial point
            ridx=IDX4j(allpj==ip(i));

            % If intenisity in original image is higher for the initial
point
            % than for the end point, we remove the endpoint
            if v0(pip(2),pip(1),pip(3))>=v0(pep(2),pep(1),pep(3))
                IDX4j(allpj==ep(j))=0;
                allpj(allpj==ep(j))=0;

            % If intenisity in original image is higher for the end point
            % than for the initial point, we remove the initial point
            else
                IDX4j(allpj==ip(i))=0;
                allpj(allpj==ip(i))=0;
            end

            % Remove 0 values from the arrays
            allpj=allpj(allpj~=0);
            IDX4j=IDX4j(IDX4j~=0);

            % Put endpoint to 0, because it has already been joined to
            % initial point
            ep(j)=0;

            % Join trajectories, always removing the initial point
tracklet
            % and replacing it with the end point tracklet
            IDX4j(IDX4j==ridx)=jidx;

            % Change joined status
            joined=1;
        end
    end
end

allj=unique(IDX4j);

% Plot
if plt4==1
    figure;
    for i=allj'
        ip1=min(allpj(IDX4j==i));

plot3(coords(allpj(IDX4j==i),1),coords(allpj(IDX4j==i),2),coords(allpj(ID
X4j==i),3),'k.-');
        hold on
```

```
        if coords(ip1,3)>1
            ip=[ip;ip1];
            cip=[cip;coords(ip1,1:3)];

plot3(coords(ip1,1),coords(ip1,2),coords(ip1,3),'g.','MarkerSize',4);
        end
        ep1=max(allpj(IDX4j==i));
        if coords(ep1,3)<25
            ep=[ep;ep1];
            cep=[cep;coords(ep1,1:3)];

plot3(coords(ep1,1),coords(ep1,2),coords(ep1,3),'r.','MarkerSize',4);
        end
    end
    view(0,0)
end

% Distinguish between moving and static, according to speed and distance
% travelled
moving6=[];
static6=[];
IDXs6=[];
IDXm6=[];

for ii=allj'
    minposx=min(coords(allpj(IDX4j==ii),1));
    maxposx=max(coords(allpj(IDX4j==ii),1));
    dist=maxposx-minposx;

    % Compute speed

    % Take all coords of the points of a tracklet
    aux=coords(ii,1:3);

    % Compute distance travelled between points
    aux=diff(aux,1,1);

    aux2=[];
    VMp=[];

    for jj=1:size(aux,1)

       % If the point hasn't been found during more than one frame, speed
       % has to be reconverted
       if(aux(jj,3)>1)

           % Divide distance travelled by number of frames during which
           % this distance has been trabelled
           gg=aux(jj,1)/aux(jj,3);

           % Add speed in speed vector
           aux2=[aux2;ones(aux(jj,3),1)*gg];

       else
           % Speed is already computed and add to the speed vector
```

```
            aux2=[aux2;aux(jj,1)];

        end

    end

    % Convert speed vector to corresponding units (microns/second)
    VMp=[VMp;aux2*DX/DT];

    % If distance travelled is smaller than 15 and max speed is low,
    % consider it a static tracklet, else consider it a moving
    if dist<=15 | max(abs(VMp))<velth
        static6=[static6;allpj(IDX4j==ii)];

        for jj=1:length(allpj(IDX4j==ii))
            IDXs6=[IDXs6;ii];
        end

    else
        moving6=[moving6;allpj(IDX4j==ii)];

        for jj=1:length(allpj(IDX4j==ii))
            IDXm6=[IDXm6;ii];
        end

    end

end

mov6=unique(IDXm6);
sta6=unique(IDXs6);

% Plot
if plt5==1
    figure;
    for i=mov6' %For each trajectory

plot3(coords(moving6(IDXm6==i),1),coords(moving6(IDXm6==i),2),coords(movi
ng6(IDXm6==i),3),'r.-');
        hold on
    end
    for i=sta6' %For each trajectory

plot3(coords(static6(IDXs6==i),1),coords(static6(IDXs6==i),2),coords(stat
ic6(IDXs6==i),3),'k.-');
        hold on
    end
    view(0,0)
end
```

Function to create results: mitoresults.m

```
function
[VM,AM,MAM,IM,VS,AS,MAS,IS,alltrack,allpoints,allindex,TA]=mitoresults(mo
v6,sta6,moving6,static6,IDXm6,IDXs6,DX,DT,coords,arees,majorax,intens)

%%%%%% OUTPUT MOVING %%%%%%%%

% Speed
VM=cell(length(mov6),1);

% Area
AM=cell(length(mov6),1);

% Major Axis Length
MAM=cell(length(mov6),1);

% Intensity
IM=cell(length(mov6),1);

for ii=1:length(mov6)
    inds=sort(moving6(IDXm6==mov6(ii)));

    % Convert area to correct units (microns squared)
    AM{ii}=arees(inds)*DX*DX;

    % Convert Major Axis Length to correct units (microns)
    MAM{ii}=majorax(inds)*DX;

    % Intensity
    IM{ii}=intens(inds);

    % Speed computation
    aux=coords(inds,1:3);
    aux=diff(aux,1,1);
    aux2=[];
    for jj=1:size(aux,1)
       if(aux(jj,3)>1)
           gg=aux(jj,1:2)/aux(jj,3);
           aux2=[aux2;ones(aux(jj,3),1)*gg];
       else
           aux2=[aux2;aux(jj,1:2)];
       end
    end

    % Convert speed to correct units (microns/second)
    VM{ii}=aux2*DX/DT;
end

%%%%%% OUTPUT STATIC %%%%%%%%

% Speed
VS=cell(length(sta6),1);

% Area
```

```
AS=cell(length(sta6),1);

% Major Axis Length
MAS=cell(length(sta6),1);

% Intensity
IS=cell(length(sta6),1);

for ii=1:length(sta6)
    inds=sort(static6(IDXs6==sta6(ii)));
    AS{ii}=arees(inds)*DX*DX;
    MAS{ii}=majorax(inds)*DX;
    IS{ii}=intens(inds);
    aux=coords(inds,1:3);
    aux=diff(aux,1,1);aux2=[];
    for jj=1:size(aux,1)
       if(aux(jj,3)>1)
           gg=aux(jj,1:2)/aux(jj,3);
           aux2=[aux2;ones(aux(jj,3),1)*gg];
       else
           aux2=[aux2;aux(jj,1:2)];
       end
    end
    VS{ii}=aux2*DX/DT;
end


%%%%%% OUTPUT OF ALL TRACKLETS %%%%%%%%

% All tracklets together, moving and static
alltrack=[mov6;sta6];

% All mitos together, moving and static
allpoints=[moving6;static6];

% All tracklets identificators together, moving and static
allindex=[IDXm6;IDXs6];

% Matrix of tracklets
TA=zeros(length(alltrack),5);

% All speeds together, moving and static
V=[VM;VS];

for i=1:length(alltrack) %For each tracklet (each row)

    % If it's a moving tracklet, replace the 0 of the first column for a
1
    if numel(intersect(alltrack(i),mov6))
        TA(i,1)=1;
    end

    % Include duration (#†of frames) in the second column
    TA(i,2)=length(allindex(allindex==alltrack(i)));
```

```
    % All points of the tracklet that we are considering
    ptraj=allpoints(allindex==alltrack(i));

    % Accumulated movement at the beginning is 0
    acc=0;

    for j=1:length(ptraj)-1 %For all points in the tracklet

        % X distance from point to point (consecutive)
        accx=abs(coords(ptraj(j),1)-coords(ptraj(j+1),1));

        % Y distance from point to point (consecutive)
        accy=abs(coords(ptraj(j),2)-coords(ptraj(j+1),2));

        % Distance module
        acc=acc+sqrt(accx^2+accy^2);

    end

    % Indicate accumulated movement on column 3
    TA(i,3)=acc;

    % Compute range of movement of each tracklet

    % Diference between minimum and maximum x position of a tracklet
    minx=min(coords(allpoints(allindex==alltrack(i)),1));
    maxx=max(coords(allpoints(allindex==alltrack(i)),1));
    dx=maxx-minx;

    % Diference between minimum and maximum y position of a tracklet
    miny=min(coords(allpoints(allindex==alltrack(i)),2));
    maxy=max(coords(allpoints(allindex==alltrack(i)),2));
    dy=maxy-miny;

    % Range of movement is placed on column 4
    TA(i,4)=sqrt(dx^2+dy^2);

    % Mean Speed
    meanv=mean(V{i,1});
    TA(i,5)=sqrt(meanv(1)^2+meanv(2)^2);
end
```

Function to create kymograph: mitokym.m

```
function
mitokym(mov6,sta6,a,b,ims,v0,expath,maxsize,IDXm6,IDXs6,moving6,static6,c
oords)
figure;
set(gcf,'position',[043    274    1688    766]);
clf;
mg=.05;
se=.2;
axes('position',[mg se+mg/2 1-2*mg 1-se-mg*1.33]);
hold on;
```

```
for i=mov6' %For each trajectory
    plot(coords(moving6(IDXm6==i),1),coords(moving6(IDXm6==i),3),'r.-');
end
for i=sta6' %For each trajectory
    plot(coords(static6(IDXs6==i),1),coords(static6(IDXs6==i),3),'k.-');
end

xlim([0,b+1]);ylim([0,length(ims)+1]);
ylabel('frames');
set(gca,'xtick',[],'xticklabel',{});
axes('position',[mg mg 1-2*mg se-mg/2]);
II=zeros(a,b,3);II(:,:,2)=v0(:,:,1);imagesc(uint8(II));
imagesc(uint8(II));
set(gca,'ytick',[],'yticklabel',{});

saveWysiwyg(gcf,[expath '/main3_' expath(end-5:end) '_' num2str(maxsize)
'kym.png']);
```

Code to find best parameters for MitoQuant: trobar_parametre_mq.m

```
fol='/Users/claudiaserrano/Documents/Claudia/Enginyeria Biomedica/Quart
2017-2018/TFG/Mito3/'; % ruta de les imatges
S=dir(fol);
SS=[];
for i=1:size(S,1)
    if contains(S(i).name,'EXP')
        SS=[SS,i];
    end
end

EQ=ones(14,7);
EQstd=ones(14,7);
for tt=1:3:20 %Threshold value
    for ts=1:2:13 %Threshold size value
        ensenya([' Starting size= ' num2str(ts) 'th=' num2str(tt)])
        EQe=ones(1,size(SS,2));
        e=0;
        for ff=[SS(1:41),SS(43:end)] %For each experiment
            e=e+1;
            %ensenya([ ' Starting ' S(ff).name]) ;
            %LASSIE
            exp=S(ff).name;
            [volSJ]=trajlassie2(exp);
            cd('/Users/claudiaserrano/Documents/Claudia/Enginyeria
Biomedica/Quart 2017-2018/TFG/MitoTrafficClaudia/Validation GT')
            %MITOQUANT
            exp1=[exp '.tif'];
            [D]=trajmq2(exp1,ts,tt);
            cd('/Users/claudiaserrano/Documents/Claudia/Enginyeria
Biomedica/Quart 2017-2018/TFG/MitoTrafficClaudia/Validation GT')
            n=ones(1,size(D,3));
            m=ones(1,size(D,3));
            for ii=1:size(volSJ,3) %for each frame
                n(ii)=numel(unique(volSJ(:,:,ii)))-1;
                m(ii)=size(D{1,ii},1);
            end
            EQe(e)=sum((n-m).^2)/size(D,3);
```

```
        end
        EQ(tt,ts)=mean(EQe);
        EQstd(tt,ts)=std(EQe);
    end
end

%figure;
%plot(1:13,EQ)
ii=0;
for i=1:3:20
    ii=ii+1;
    jj=0;
    for j=1:2:13
        jj=jj+1;
        EQs(ii,jj)=EQ(i,j);
    end
end
```

GUI code

```
function varargout = prova2(varargin)
% prova2 MATLAB code for prova2.fig
%      prova2, by itself, creates a new prova2 or raises the existing
%      singleton*.
%
%      H = prova2 returns the handle to a new prova2 or the handle to
%      the existing singleton*.
%
%      prova2('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in prova2.M with the given input
arguments.
%
%      prova2('Property','Value',...) creates a new prova2 or raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before prova2_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to prova2_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help prova2

% Last Modified by GUIDE v2.5 13-Feb-2018 11:19:33

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @prova2_OpeningFcn, ...
                   'gui_OutputFcn',  @prova2_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
```

```
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before prova2 is made visible.
function prova2_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to prova2 (see VARARGIN)

% Choose default command line output for prova2
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes prova2 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = prova2_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



function edit1_Callback(hObject, eventdata, handles)
data = guidata(hObject);
% Get location of files
value = get(handles.edit1,'String');
if numel(value)<3
    miss=3-numel(value);
    add=zeros(1,miss);
    t1=num2str(add);
    t2=value;
    nfold0=[t1 t2];
else
    nfold0=value;
```

```
end
nfold= nfold0(~isspace(nfold0));
data.fold=[data.dir '/EXP' nfold];
data.nfold=nfold;

% Get files (load volum)
expath=data.fold;
tag='ch00';
ims=[dir([expath '/*' tag '*.tif']) dir([expath '/*' tag '*.jpg'])];
[I,map]=imread([expath '/' ims(1).name]);
[a,b,c]=size(I);
ch=1;
if(c>1)

[~,ch]=max([sum(sum(I(:,:,1))),sum(sum(I(:,:,2))),sum(sum(I(:,:,3)))]);
end
volum=zeros(a,b,length(ims));
for ii=1:length(ims)
    I=double(imread([expath '/' ims(ii).name]));
    volum(:,:,ii)=I(:,:,ch);
end
v0=volum;
data.ims=ims;


% Display initial image
s=size(v0);
v=zeros([s(1:2),3,25]);
for i=1:s(3)
    v(:,:,1,i)=zeros(s(1:2));
    v(:,:,2,i)=v0(:,:,i);
    v(:,:,3,i)=zeros(s(1:2));
end
v=v/256;
data.v=v;
imagesc(v(:,:,:,1))
set(handles.axes1,'Visible','off');

%Modify slider
set(handles.slider1,'Min',1);
set(handles.slider1,'Max',length(ims));
set(handles.slider1,'Value',1)
set(handles.slider1,'SliderStep',[1/(length(ims)-1) 1/(length(ims)-1)]);

% Reset tracks
data.IT = 0;
data.xTrack=[];
data.yTrack=[];
set(handles.listbox1,'String',{});
data.state=0;
data.cV=1;
map=[map;1 0 1];
data.map=map;
set(handles.edit2,'String',num2str(data.cV));
Mark=cell(6,2);
Mark{1,1}='m';Mark{1,2}='+';
Mark{2,1}='m';Mark{2,2}='o';
```

```
Mark{3,1}='r';Mark{3,2}='+';
Mark{4,1}='r';Mark{4,2}='o';
Mark{5,1}='y';Mark{5,2}='+';
Mark{6,1}='y';Mark{6,2}='o';
data.Mark=Mark;

guidata(hObject,data);
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a
double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
data = guidata(hObject);
data.dir=uigetdir;
guidata(hObject,data);
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
data = guidata(hObject);
data.IT = data.IT + 1; % Update track #
itadd = num2str(data.IT);
contents = cellstr(get(handles.listbox1,'String')); %Actual (without
adding) contents of the listbox
list = [contents;itadd];
set(handles.listbox1,'String',list);
data.list=list;
data.xTrack=[data.xTrack;zeros(1,length(data.ims))];
data.yTrack=[data.yTrack;zeros(1,length(data.ims))];
guidata(hObject,data);
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
data = guidata(hObject);
if data.IT>0
    data.list{end}={};
    data.list=data.list(~cellfun('isempty',data.list));
    data.IT = data.IT - 1;
    set(handles.listbox1,'String',data.list);
    data.xTrack=data.xTrack(1:end-1,:);
    data.yTrack=data.yTrack(1:end-1,:);
end
guidata(hObject,data)
%contents = cellstr(get(hObject,'String'))
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton4. %START
function pushbutton4_Callback(hObject, eventdata, handles)
data = guidata(hObject);
data.state = 1;
data.lastv=data.v;
data.lastx=data.xTrack;
data.lasty=data.yTrack;
[x,y]=getpts;
contents = cellstr(get(hObject,'String'));
row=contents{get(hObject,'Value')};
data.xTrack(data.IT,data.cV)=x;
data.yTrack(data.IT,data.cV)=y;
i=data.cV;
v=data.v;
Mark=data.Mark;
p1=get(handles.listbox1,'Value');
RGB = insertMarker(v(:,:,:,i),[x,y],Mark{p1,2},'color',Mark{p1,1});
imagesc(RGB)
v(:,:,:,i)=RGB;
set(handles.axes1,'Visible','off');
data.v=v;

guidata(hObject,data);
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on selection change in listbox1.
function listbox1_Callback(hObject, eventdata, handles)

% hObject    handle to listbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: contents = cellstr(get(hObject,'String')) returns listbox1
contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
listbox1


% --- Executes during object creation, after setting all properties.
function listbox1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to listbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on slider movement.
function slider1_Callback(hObject, eventdata, handles)
data = guidata(hObject);
cV = round(get(hObject,'Value'));
v=data.v;
imagesc(v(:,:,:,cV))
set(handles.axes1,'Visible','off');
set(handles.edit2,'String',num2str(cV));
data.cV=cV;
if data.state==1
    data.lastv=data.v;
    data.lastx=data.xTrack;
    data.lasty=data.yTrack;
    [x,y]=getpts;
    data.xTrack(data.IT,data.cV)=x;
    data.yTrack(data.IT,data.cV)=y;
    i=data.cV;
    v=data.v;
    Mark=data.Mark;
    p1=get(handles.listbox1,'Value');
    RGB = insertMarker(v(:,:,:,i),[x,y],Mark{p1,2},'color',Mark{p1,1});
    imagesc(RGB)
    v(:,:,:,i)=RGB;
    data.v=v;
    set(handles.axes1,'Visible','off');
    set(handles.edit2,'String',num2str(cV));
end
guidata(hObject,data);
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of
slider
```

```
% --- Executes during object creation, after setting all properties.
function slider1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
data = guidata(hObject);
data.state=0;
guidata(hObject,data);
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as a
double


% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
data = guidata(hObject);
```

```
nfile=['EXP' data.nfold '.mat'];
[file,path] = uiputfile(nfile,'Save file name');
cd (path);

xTrack=data.xTrack;
yTrack=data.yTrack;
save(file,'xTrack','yTrack');

% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton7. UNDO
function pushbutton7_Callback(hObject, eventdata, handles)
data = guidata(hObject);
data.v=data.lastv;
data.xTrack=data.lastx;
data.yTrack=data.lasty;
i=data.cV;
v=data.v;
imagesc(v(:,:,:,i))
set(handles.axes1,'Visible','off');
guidata(hObject,data);
% hObject    handle to pushbutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

Code to compute track based error: track_based_error.m

```
%trajlassie: 'EXP032'
%trajmq: 'EXP032.tif'
%GT: 'EXP032.mat'

expath='/Users/claudiaserrano/Documents/Claudia/Enginyeria
Biomedica/Quart 2017-2018/TFG/Proves App/prova save';

files=dir([expath '/*' 'EXP*.mat']);
Tt=0; %True track
Tcmq=[]; %Ytracked MitoQuant
Tcl=[]; %Ytracked Lassie
tdL=0; %Number of true tracks detected by Lassie
tdM=0; %Number of true tracks detected by MitoQuant
for ff=1:size(files) %For each experiment tracked
    cd('/Users/claudiaserrano/Documents/Claudia/Enginyeria
Biomedica/Quart 2017-2018/TFG/MitoTrafficClaudia/Validation GT')
    ensenya([ ' Starting ' files(ff).name]) ;
    cd (expath)
    load(files(ff).name);
    for tt=1:size(xTrack,1) %For each true trajectory in the experiment
        xlt=[];
        Tt=Tt+1;
        xTrackk=xTrack(tt,:); %Read x-coords of true track
        yTrackk=yTrack(tt,:); %Read y-coords of true track
        ii=find(xTrackk>0); %Find only the tracked points
```

```
        xG=ii; %Tracked points

        %LASSIE
        fold0=extractBefore(files(ff).name,".mat");
        cd ('/Users/claudiaserrano/Documents/Claudia/Enginyeria
Biomedica/Quart 2017-2018/TFG/MitoTrafficClaudia/Validation GT')
        [coords,allindex,allpoints]=trajlassie(fold0); %Run Lassie
program
        xL=[];
        tL=[];
        for i=find(xTrackk>0) %For each point of the GT track
            %Those points inside a given radius to the GT track point
            p=find(coords(:,1)>xTrackk(i)-12 & coords(:,1)<xTrackk(i)+12
& coords(:,3)==i & coords(:,2)>yTrackk(i)-6 & coords(:,2)<yTrackk(i)+6);
            if length(p)>1 %If a point has been found
                distt=[];
                for j=1:length(p) %For all points found, compute the
relative distance to point
                distt=[distt;sqrt((coords(p(j),1)-
xTrackk(i))^2+(coords(p(j),2)-yTrackk(i))^2)];
                end
                [a,pp]=min(distt); %Select the point with minimum
distance to GT point
                p=p(pp);
            end
            if ~isempty(p) %If a point has been found
                traL=allindex(allpoints==p); %Find track number
                if ~isempty(traL)
                    tL=[tL;traL]; %Save track number
                    xL=[xL;p]; %Save point number
                end
            end
        end

        t1L=unique(tL);
        Ncounts=histc(tL,t1L);
        t2L=t1L(Ncounts>=3); %Only save those tracks with 3 or more
points correctly detected

%         for i=t2L'
%
plot3(coords(allpoints(allindex==i),1),coords(allpoints(allindex==i),2),c
oords(allpoints(allindex==i),3),'k.-');
%         end

        if ~isempty(t2L)
            xlt=xL(logical(sum(tL'==t2L,1)')); % Points correctly
detected
            tdL=tdL+1;%Number of detected true tracks

%plot3(coords(xlt,1),coords(xlt,2),coords(xlt,3),'ko','MarkerSize',6)
        end

        %MITOQUANT
        xmt=[];
        fold=[fold0 '.tif'];
        cd ('/Users/claudiaserrano/Documents/Claudia/Enginyeria
```

```
Biomedica/Quart 2017-2018/TFG/MitoTrafficClaudia/Validation GT')
        [T2]=trajmq(fold);
        xM=[];
        tM=[];
        for i=find(xTrackk>0) %For each point of the GT track
            p=find(T2.x(:,i)>xTrackk(i)-12 & T2.x(:,i)<xTrackk(i)+12 &
T2.y(:,i)>yTrackk(i)-8 & T2.y(:,i)<yTrackk(i)+6);
            if length(p)>1
                distt=[];
                for j=1:length(p)
                distt=[distt;sqrt((T2.x(p(j),i)-
xTrackk(i))^2+(T2.y(p(j),i)-yTrackk(i))^2)];
                end
                [~,pp]=min(distt);
                p=p(pp);
            end
            if ~isempty(p)
                tM=[tM;p];
                pos=sub2ind(size(T2.x),p,i);
                xM=[xM;pos];
            end
        end

        t1M=unique(tM);
        Ncounts=histc(tM,t1M);
        t2M=t1M(Ncounts>=3);

%         for ktraj=t2M'
%             pos=[];
%             for kframe=1:size(T2.x,2)
%                 if T2.x(ktraj,kframe)>0
%                     pos=[pos;T2.x(ktraj,kframe) T2.y(ktraj,kframe)
kframe];
%                 end
%             end
%             plot3(pos(:,1),pos(:,2),pos(:,3),'g.-')
%         end

        if ~isempty(t2M)
            xmt=xM(logical(sum(tM'==t2M,1)'));
            [~,fr]=ind2sub(size(T2.x),xmt);
            tdM=tdM+1;
            %plot3(T2.x(xmt),T2.y(xmt),fr,'go','MarkerSize',6)
        end
        Tcmq=[Tcmq;length(xmt)/length(xG)];
        Tcl=[Tcl;length(xlt)/length(xG)];
    end
end

Etmq=1-(sum(Tcmq)/Tt);
Etl=1-(sum(Tcl)/Tt);

mL=mean(Tcl);
mMQ=mean(Tcmq);

stdL=std(Tcl);
```

```
stdMQ=std(Tcmq);

seL=stdL/sqrt(length(Tcl));
seMQ=stdMQ/sqrt(length(Tcmq));

%We want Error (Etmq, Etl) to be small, the smaller the error, the better
%the code
```